Høgskolen i Østfold

MASTEROPPGAVE

Håndtering av tid og hendelsesforløp i sluttbrukerverktøy for scenariobaserte virtuelle simuleringer

Bjørn Arild Lunde

Dato: 15.12.2021

Masters in Applied Computer Science
Avdeling for informasjonsteknologi

Takk til

Jeg ønsker først og fremst å takke min veileder Joakim Karlsen for hans motiverende og konstruktive tilbakemeldinger gjennom hele prosjektet. Han viser tydelig at han brenner for forskning og har en smittende arbeidsglød. Jeg tror ikke jeg hadde klart å fullføre prosjektet uten han. Jeg ønsker også å rette en takk til Inger Hjelmeland og Marian Arntsen fra Høgskolen i Østfold sin avdeling for Helse og Velferd i Fredrikstad som hjalp meg å rekruttere kolleger og studenter til datainnsamlingen min. Jeg vil også si takk til Ann-Charlott Karlsen, Fahad Faisal Said, Stefan Christiansen og Kristina Mormil Protrka som hjalp meg med korrekturlesing. Sist vil jeg gi en stor takk til mamma, pappa, min søster Anne Sofie og min samboer Silje som har motivert meg, heiet på meg og gitt meg tid og rom for å fullføre oppgaven.

Sammendrag

Helsesektoren er et av de fagfeltene forskere mener kunne ha mest nytte av digitale opplæringsverktøy for å trene sine ansatte i trygge miljøer hvor det er mulig å gjøre feil uten at det får kritiske følger. Virtuelle simuleringer og serious games for opplæring av helsepersonell eksisterer i dag, men etter hvert som det blir gjort gjennombrudd i forskning og prosedyrer forandres blir disse verktøyene både dyre og tidskrevende å vedlikeholde. En mulig løsning på dette er å utvikle sluttbrukerverktøy for denne målgruppen som gir dem muligheten til å selv kunne manipulere og oppdatere opplæringsverktøyene sine. Dette masterprosjektet vil derfor se på hvordan vi kan gi helsepersonell muligheten til å håndtere tid og hendelsesforløp i virtuelle simuleringer ved hjelp av visuell programmering. De to mest populære formene for visuell programmering i dag er nodebasert og blokkbasert programmering. Problemstillingen i masteroppgaven ønsker å utforske hvilken av de to tilnærmingene til visuell programmering som best balanserer kompleksitet og brukervennlighet for instruktører som har i oppgave å lage scenariobaserte virtuelle simuleringer for opplæring av helsepersonell. For å kunne svare på forskningsspørsmålet har det derfor blitt gjennomført en kvalitativ studie hvor prosjektarbeidet har blitt delt opp i fire faser: Innsikt, idéutvikling, utvikling og testing. Utredning av resultater fra datainnsamlingen indikerer at en nodebasert tilnærming til visuell programmering er det som best balanserer kompleksitet og brukervennlighet for målgruppen til dette prosjektet. Resultatet av oppgaven vil derfor være retningslinjer og anbefalinger for videre design og utvikling av et visuelt programmeringsgrensesnitt som benytter seg av nodebasert programmering.

Innholdsfortegnelse

Figurer og tabeller

1 Introduksjon

Vi lever i en tid hvor teknologi er en del av hverdagen og samfunnet stadig blir mer digitalisert. Måten vi kommuniserer, måten vi samler informasjon på og måten vi ser på verden blir i stor grad påvirket av teknologien vi omgir oss med. Dette er også tilfellet for hvordan vi lærer, enten det er i skole eller jobb. Helsesektoren er et av fagfeltene som står mest sentralt når det kommer til utvikling av teknologi for å støtte opplæring(Sharifzadeh et al., 2020), og en av teknologiene som viser seg å ha stort potensiale er virtuelle simuleringer. Helse og medisin er som alle andre fagfelt under stadig utvikling, noe som er positivt med tanke på folks helse og velvære, men som kan være utfordrende med tanke på opplæring. Ting i dag blir ikke nødvendigvis gjort på samme måte som for 10 år siden, og opplæringsverktøy og materiale må oppdateres etter nye gjennombrudd i forskningen. I dette tilfellet vil det være naturlig å hente inn personell, enten internt eller eksternt, som kan tilpasse verktøyene til endringene. Jeg tror at man kan spare verdifulle ressurser ved å gi sluttbrukerne selv mulighetene til å gjøre disse endringene. I sammenheng med dette prosjektet vil sluttbrukerne være instruktører som har i oppgave å lage scenariobaserte virtuelle simuleringer, men begrepet sluttbruker kan i all sin enkelhet ses på som en person om bruker en datamaskin (Ko et al., 2011).

En virtuell simulering er en lokasjon og et miljø som blir gjenskapt av en datamaskin og prosjektert på en digital flate (Padilha et al., 2019) slik som VR-briller eller lignende. En simulering som dette kan også bestå av karakterer og en tidslinje med forskjellige elementer som forteller en historie eller et hendelsesforløp for å gjøre simuleringer mer realistisk og tiltalende. For å ha ferdighetene til å kunne skrive kode og manipulere komplekse entiteter slik som virtuelle simuleringer krever dette en forståelse for programmering i tillegg til de reglene og prinsippene som er en del av fagfeltet. En motiverende faktor til dette prosjektet er å kunne senke denne kunnskapsbarrieren ved å pakke inn tekstbasert programmering i et grafisk grensesnitt som skal være brukervennlig selv for sluttbrukere som ikke har noen forkunnskaper om programmering. Det finnes allerede flere slike visuelle programmeringsspråk, hvor de mest populære er blokkbasert programmering og nodebasert programmering. Jeg ønsker å grave dypere i begge disse tilnærmingene til visuell programmering for å finne ut hvilken av dem best balanserer brukervennlighet og kompleksitet i sammenheng med utvikling av verktøy for sluttbrukere som skal brukes til å tilpasse scenarioer i opplæringsverktøy.

Et konsept som muligens kan senke terskelen for å kunne forstå og skrive programkode er direkte manipulasjon. Dette er et konsept som baserer seg på at brukergrensesnittet er bygget opp med visuelle elementer som sluttbrukere kan peke og klikke på for å gi de en responsiv og brukervennlig opplevelse (Shneiderman, 1997). Schneiderman mener at det er tre punkter som definerer direkte manipulasjon:

1. Kontinuerlig visuell representasjon av objekter.
2. Alle handlinger innebærer at man kan trykke på knapper istedenfor å benytte seg av syntaks.
3. Operasjoner skal være mulig å reversere raskt og enkelt.

Det første dokumenterte forsøket på direkte manipulasjon kan spores tilbake til så tidlig som på 60-tallet (Hutchins et al., 1985), og har banet vei for grafiske grensesnitt slik vi kjenner dem i dag.

1.1 Forskningsspørsmål

Formålet med oppgaven er som nevnt innledningsvis å utforske hvordan instruktører i helsesektoren kan håndtere tid og hendelsesforløp i opplæringsverktøyene sine uten hjelp av eksternt personell. Visuell programmering kan trolig gi sluttbrukerne denne muligheten, men da helsepersonell ikke har noen formell opplæring i programmering må verktøyene også tilpasses etter deres forutsetninger. Forskningsspørsmålet blir derfor som følger:

Hvilken av de to mest populære visuelle programmeringsgrensesnittene (blokkbasert og nodebasert) balanserer best kompleksitet og brukervennlighet for instruktører som har i oppgave å lage scenariobaserte virtuelle simuleringer som brukes til opplæring av helsepersonell?

Resultatet av prosjektet vil være et sett med anbefalinger og retningslinjer for videre utvikling av et visuelt programmeringsgrensesnitt basert på funn fra litteraturen og en diskusjon av innsamlet data. For å legge frem disse resultatene har det først blitt gjort et litteratursøk som tar for seg aktuelle temaer som sluttbrukerverktøy, utfordringer med å lære programmering og tidligere prosjekter som har brukt visuell programmering i sluttbrukerverktøy i andre domener. Basert på informasjonen som finnes i litteraturen vil kapittel 2, som tar for seg metode, beskrive de fire fasene av prosjektarbeidet som til slutt endte med en kvalitativ

datainnsamling. Kapittel 3 inneholder en analyse og sammenligning av den innsamlede dataen som til slutt ender med et svar på forskningsspørsmålet. Til slutt så dekkes diskusjon, konklusjon og anbefalinger for videre arbeid i kapittel 4.

1.2 Litteraturgjennomgang

I dette kapittelet presenteres relevant litteratur som legger grunnlaget for produksjon, diskusjon og analyse av datainnsamlingen som kommer i senere kapitler. Sluttbrukerverktøy og det å lære seg programmering kommer ikke uten utfordringer, og begge disse temaene er diskutert i litteraturen. Videre er domene ekspert et naturlig begrep å definere i forhold til sluttbrukerverktøy, hva dette er og hva dette betyr i sammenheng med dette prosjektet. Da hovedmålet med oppgaven er å utforske to forskjellige tilnærminger til visuell programmering er det hensiktsmessig å se hvordan forskjellene på blokkbasert og nodebasert programmering blir definert. Avslutningsvis vil eksisterende verktøy som benytter seg av visuell programmering og lignende verktøy i andre domener utforskes.

1.2.1 Hva er et sluttbrukerverktøy?

Det er vanskelig å forutse kunnskapsutvikling og tilpasse opplæringsverktøy for endringer som enda ikke er utforsket. Det er høyst sannsynlig at prosedyrer slik vi kjenner dem i dag på et eller annet tidspunkt vil bli endret. Det å gi sluttbrukerne muligheten til å kunne tilpasse innhold selv uten assistanse fra profesjonelle utviklere er den generelle ideen bak sluttbrukerverktøy (Menestrina & De Angeli, 2017). Et sluttbrukerverktøy kan også ses på som et verktøy som gir sluttbrukerne mulighet til å selv lage programmer med et programmeringsspråk utviklet spesifikt for denne målgruppen (Barricelli et al., 2019). I mange yrker i dag brukes det programvare med forskjellig funksjonalitet som hjelper til med diverse ting som må gjøres gjennom en arbeidsdag. Fischer påstår at sluttbrukerverktøy er helt nødvendig for at man ikke skal bli sittende fast i gamle rutiner som følge av programvare som ikke blir oppdatert (Fischer, 2009). I tillegg til å holde ansatte oppdaterte kan også sluttbrukerverktøy føre til lavere kostnader da endringene kan gjøres av interne sluttbrukere istedenfor utviklere. Likevel er det noen utfordringer med slike løsninger som viser seg å være gjentagende. Det er viktig å huske at sluttbrukerne i mange tilfeller ikke har hverken fagkunnskaper eller forståelse av utviklingsprosessen og vedlikehold som det profesjonelle utviklere har (M. F. Costabile et al., 2008). Costabile et al. (2008) deler opp utviklere og sluttbrukere i to forskjellige grupper; Eierne av problemet og eierne av teknologien. Videre

forklarer forfatterne at disse to gruppene også har sitt eget «språk» knyttet opp mot sitt yrke, og at det er viktig at de omfavner motpartens slang og terminologi for å kunne dele kunnskapen sin på en fornuftig måte. Dette betyr for eksempel at sluttbrukerne må sette seg inn i «språket» til utviklerne for å kunne håndtere verktøyene de får tildelt, noe som for mange er demotiverende og avhenger av hvor villig hvert enkelt individ er til å lære. En måte å gjøre denne språkbarrieren mindre på kan være å inkludere sluttbrukerne i designprosessen under produksjonen av verktøyet slik at de kan være med å ta avgjørelser for design og funksjonalitet underveis.

Meta design er et konseptuelt rammeverk for å forme nye måter å gjøre designprosessen til en prosess der samarbeid står sentralt (Fischer & Giaccardi, 2006). Denne prosessen inviterer sluttbrukerne til å ta en del i designprosessen og ikke bare under utviklingen, men under hele livssyklusen til et produkt. På denne måten kan de komme med innspill og ideer som passer deres behov. Meta design setter også designprosessen i førersetet, og denne delen er sett på som like viktig som produksjonen av selve simuleringen (Fischer et al., 2004). Noen ganger krever problemer mer enn et hode for å forstå utfordringen og finne løsninger. Ved å slå sammen de to gruppene til en gruppe som jobber sammen, så kan man se på problemet fra et nytt ståsted og det kan takles fra flere vinkler man tidligere ikke kunne se. Fischer og Giaccardi (2006) forklarer at kunnskapskravet knyttet til å løse problemer ikke nødvendigvis kommer tydelig frem før man plukker problemet fra hverandre bit for bit og nettopp dette er en av grunnene til at denne prosessen med samarbeid er så effektiv da man underveis i designfasen kan få evalueringer av de som faktisk skal bruke verktøyet.

1.2.2 Utfordringer med sluttbrukerverktøy

Selv om ideen bak sluttbrukerverktøy er god så kommer det ikke uten utfordringer. Et vanlig problem med sluttbrukerverktøy for manipulasjon av virtuelle simuleringer som blir brukt til opplæring og trening av ansatte er å finne balansen mellom kompleksiteten av verktøyet og brukervennlighet. Med andre ord, de verktøyene som er enkle å bruke mangler fleksibiliteten og funksjonaliteten til de mer kompliserte verktøyene. Problemet med disse kompliserte verktøyene er at de ofte krever at sluttbrukerne aktivt må sette av tid og gå inn for å opparbeide seg ny kunnskap for å bruke de funksjonene det tilbyr (Menestrina et al., 2014). Menestrina et al. (2014) trekker sammenligninger til modifikasjonsverktøy i videospill, hvor sluttbrukerne har muligheten til å forme miljøer og kunstig intelligens etter deres ønsker med verktøy som Unreal Development Kit og Crysis Sandbox, som begge krever en forståelse for

programmering. Dette kravet til forkunnskaper gjør ofte at de som bruker verktøy som disse uten denne kunnskapen ofte gjør mange tekniske feil uten at de selv er klar over det (M. F. Costabile et al., 2008). På den andre siden har vi de enklere verktøyene som for eksempel modifiseringsverktøyene i spill som Minecraft, hvor man enkelt kan legge til nye elementer i den virtuelle verdenen og endre utseende på elementer i spillet slik man ønsker det. Likevel, på grunn av verktøyets enkelhet og brukervennlighet er det restriksjoner for å manipulere ting som kunstig intelligens, lage «custom» menyer eller endre spillfigurens egenskaper som hvor fort de kan løpe (Menestrina & De Angeli, 2017).

Sluttbrukerne kommer ofte fra forskjellige kulturer, er i forskjellige aldersgrupper og i vårt tilfelle kommer de fra en arbeidsgruppe som ikke krever kunnskap om utvikling av programvare til å kunne produsere virtuelle simuleringer på egenhånd. Som Costabile et al. (2007) sier så er målet å kunne tilby sluttbrukere de verktøyene som gir de den funksjonaliteten som de trenger samtidig som det hele er presentert med en innpakning og et språk som er forståelig og effektivt.

For enkelte profesjoner og arbeidsområder kan det være lite hensiktsmessig å utvikle verktøy for sluttbrukerne. Dette gjelder spesielt i felt som er brede og stadig i utvikling, Fischer et al. (2004) mener at de mindre komplekse domenene er mer passende for sluttbrukerverktøy laget for opplæring. Dette er ikke på grunn av programvaren i seg selv, men heller tiden sluttbrukerne må bruke på å tilpasse programvaren sin i forhold til hvor mye tid de faktisk får brukt det med de nyeste endringene og oppdateringene. I felt med mindre utvikling vil det være mulighet for å bruke verktøyene over lenger tid uten at de krever endringer istedenfor at all tiden vil gå med på å tilpasse verktøyet etter de nye reglene.

Å gi sluttbrukere et verktøy som kan oppnå det de ønsker er det alle vil ha, men det er også mulig å gi sluttbrukerne for mye frihet, noe som fører til et annet problem, nemlig det såkalte «Turing tar pit». Dette fenomenet beskrives som en situasjon hvor alt er mulig, men ikke noe fornuftig er enkelt (Cunningham et al., 2021). Det har med andre ord ikke noen hensikt å lage et verktøy som lar sluttbrukerne gjøre alt dersom de ikke klarer å bruke det effektivt.

1.2.3 Utfordringer med å lære programmering

Som beskrevet i 1.2.1 så er ofte programmeringsgrensesnitt i sluttbrukerverktøy laget spesifikt for sluttbrukerne. Et slikt språk kalles et domene spesifikt visuelt språk (DSLV). Et DSVL tilbyr et visuelt programmeringsgrensesnitt som inneholder hovedsakelig terminologi og konsepter som brukes av denne målgruppen (Sprinkle & Karsai, 2004), og dette er en egenskap som også kan finnes i tradisjonell tekstbasert programmering. Som Alan Blackwell et al. (2019) sier så er det ingen universell programmeringsløsning som er best egnet for all bruk, men at presentasjonen av grensesnittet må ta høyde for problemstillingene det skal løse.

En utfordring nye programmere står ovenfor er at mange kan bli forvirret av at det samme innholdet kan bli presentert på forskjellige nivåer. Tanaka-Ishii (2010) illustrerer denne problemstillingen med følgende kodeeksempel:

```
int x = 15
```

I dette eksempelet er innholdet (tallet 15) representert av tre forskjellige tegn. Den første er ganske åpenbart tallet i seg selv, 15. Denne verdien blir så representert av `x`, som peker på hvor denne verdien blir lagret. Sist har vi `int`, som representerer datatypen av verdien. Dette er et hett tema innenfor design av programmeringsspråk og utfordringen for nykommere i følge Tanaka-Ishii er at det kan være forvirrende hvordan disse tre tegnene tolkes. For en som har noe erfaring med programmering vil vedkommende fort kunne identifisere hva disse tegnene betyr og hvordan informasjonen kan tas i bruk da dette er definert i spesifikasjonene og syntaksen til språket.

Kumiko Tanaka-Ishii beskriver også at det ofte blir trukket paralleller mellom mennesker og datamaskiner i det dagligdagse liv, science fiction og filosofiske diskusjoner, noe hun mener ofte fører til konklusjonen om at språkene våre til dels har likheter (Tanaka-Ishii, 2010). Likevel, selv etter så mange år med forskning som ligger bak semiotikken og måten vi kommuniserer med datamaskiner på, så er det utfordringer som gjør at mange ikke klarer å kommunisere med en maskin slik som de ønsker. Tanaka-Ishii foreslår at dette kan være fordi både kalkulasjonene maskinene gjør og naturlig språk som vi mennesker benytter oss av i bunn og grunn er prosessering av tegn, men at utformingen og presentasjonen av disse tegnene er så forskjellig at dette medfører at mennesker ikke forstår maskiner så godt.

En annen utfordring nye programmere kan kjenne seg igjen i er som Alan Blackwell forklarer en problemstilling som han referer til som «the frame problem». Dette problemet oppstår når det ikke er satt noen klar grense for konsekvensene av en handling, med andre ord, bruken av funksjonalitet kan ha uendelige utfall (Blackwell, 2002). For å unngå dette problemet burde det settes grenser på omfanget av funksjonalitet slik at hvert enkelt element har en egen oppgave. Noe som også kan være utfordrende for nye programmere er at verktøyene et språk tilbyr kan ha flere bruksområder avhengig av programmet man jobber med.

1.2.4 Hvem utvikler vi for – domene ekspert

Sluttbrukerverktøy med sine fordeler og ulemper er ikke nødvendigvis en statisk entitet man kan gjenbruke igjen og igjen. Som nevnt i foregående kapittel er det ingen programmeringsløsning som kan løse alle problemer, og det sammen gjelder for sluttbrukerverktøy. Nøkkelordet i denne sammenhengen er sluttbrukeren, hvem de er og hvordan de skal bruke verktøyene sine. For å definere egenskapene til sluttbrukerne bør man derfor utforske hva en domene ekspert er.

Det som gjør domene ekspertene til en så sentral rolle når det kommer til EUD kommer av flere grunner. For det første er domene ekspertene de som til slutt kommer til å bruke verktøyet som blir laget for dem, i vårt tilfelle et visuelt programmeringsgrensesnitt. For det andre, så er domene ekspertene de som kan ses på som «eieren av problemet» (Fischer et al., 2009). Istedenfor å kun bruke utviklerne med sin overfladiske forståelse av den domene spesifikke terminologien og problemstillingene som skal løses som informasjonskilde, så er det mye nyttig informasjon å hente direkte fra domene ekspertene. Mange sluttbrukere har også vaner i sammenheng med deres jobb, enten bevisst eller ubevisst. Disse vanene burde bli vurdert under designfasen dersom dette er mulig (M. F. Costabile et al., 2006).

En domene ekspert er en person som regnes som en ekspert innenfor deres arbeidsfelt, et felt som ikke nødvendigvis trenger å være informasjonsteknologi, og i mange tilfeller er ikke disse ekspertene heller interesserte i informasjonsteknologi (M.-F. Costabile et al., 2003). Domene ekspertene har deres egne arbeidsrutiner og språk de bruker for å kommunisere med sine kolleger gjennom for eksempel dokumenter som er strukturert på en måte som de fleste med deres kompetanse kan forstå. En av utfordringene med å utvikle sluttbrukerverktøy som er rettet direkte mot enkelte domene eksperter er at dette språket og måten de kommuniserer på og metodikken den bruker stadig er i endring. Costabile et al. sier i deres artikkel at «å

bruke systemet endrer brukeren, og etterhvert som brukerne endrer seg vil de også finne nye måter å bruke systemet på», som er et fenomen innenfor Human-Computer Interaction(HCI) som har blitt navngitt co-evolusjon (M.-F. Costabile et al., 2003). Det er to grunner til at dette fenomenet oppstår: brukerne oppdager nye behov og krav som ikke var planlagt under designfasen, og finner nye måter å bruke systemet til å dekke behovene deres. Forandringer gjennom livssyklusen til programvare skjer ikke utelukkende i sluttbrukerverktøy, flere applikasjoner som blir levert i dag har generiske og mangelfulle løsninger ved levering og utvikles videre etter hvert som applikasjonen blir brukt (Mørch, 1997).

Ko et al. (2011) beskriver et veldig tydelig skille på profesjonelle programmere og «sluttbruker-programmerere» som de velger å kalle det, som i vårt tilfelle vil bestå av domene eksperter. De sier at målet til en profesjonell programmerer er å lage et program som skal leveres og vedlikeholdes til en bestemt bruk over en lengre tid. Sluttbrukerne derimot har et mål om å lage programmer som kan gjøre aktiviteter og oppgaver innenfor deres fagområde enklere og mer effektivt (Ko et al., 2011).

1.2.5 Læringsutbytte i digitale opplæringsverktøy

Selv om sluttbrukerverktøy for virtuelle simuleringer er et komplekst fagområde med mange utfordringer slik som litteraturen har vist frem til nå er det likevel en gevinst med praktiske verktøy slik som virtuelle simuleringer i form av et godt læringsutbytte. Det er fortsatt faktorer som regulerer hvor stort dette læringsutbyttet er.

Luke Rogers mener at man bare husker 10% av informasjonen man leser, mens du sitter igjen med hele 90% av informasjonen du får gjennom praktisk læring (Rogers, 2011). I enkelte yrker er det ikke unormalt at man kan gjøre feil i jobbsammenheng og lære av disse feilene. Handlinger kan reverseres og ingen trenger nødvendigvis å bli berørt av feilene dine. Selv om dette er tilfellet i mange yrker, så er det ikke alle som har like mye spillerom til å gjøre feil, spesielt dersom du jobber med mennesker og deres helse. Å tilby folk som jobber i helsesektoren et miljø hvor de kan trene uten å sette noen sin helse på spill slik at de kan teste kunnskapen sin og lære i situasjoner som vanligvis ville vært risikable kan potensielt forhindre mange av disse feilene. I de siste årene har serious games og virtuelle simuleringer blitt sett på som potensielle kandidater til å fylle dette tomrommet som i dag er i opplæringsverktøy, og begge har gjennom eksperimenter vist seg å være lovende plattformer (Foronda et al., 2016; King et al., 2018; Westera, 2017). Sharifzadeh et al. (2020) poengterer

også at helsesektoren er et av de fagområdene som er mest aktuelle for å ta i bruk serious games i opplæring, og at praktikantene innenfor feltet er interessert i læringsformen serious games tilbyr da dette er en form for praktisk læring.

Mens vanlige videospill er laget for underholdning er serious games laget hovedsakelig for å opplyse. Spill har en evne til å holde folks oppmerksomhet over lengre tidsperioder, spesielt for de som allerede er interessert i videospill fra sitt personlige liv. Det er likevel faktorer som avgjør hvor mye man kan lære mens man spiller (Westera, 2017). Hvor attraktivt et spill er og hvor intelligente spillerne er, er to punkter som Wim Westera påstår har en positiv effekt på læringsutfallet, mens han også mener at bare fordi et spill er mer komplekst betyr ikke dette nødvendigvis at man lærer mer, snarere tvert imot. Noe som gjør serious games vanskelig å utvikle at man hele tiden jobber mot å lage noe som er både motiverende og samtidig som det skal tilegne brukeren ny kunnskap eller friske opp allerede eksisterende kunnskap. Det finnes flere eksempler på spill slik som dette som hadde både store investorer og mange timer med arbeid bak seg, men som til slutt ikke klarte å oppnå det de ønsket da kvaliteten på selve spillmekanikkene ikke holdt mål, spillerne manglet altså motivasjon til å spille (Drummond et al., 2017). Et av disse eksemplene er *Arden: The World of Shakespeare* som ble en stor flopp som fikk spillerne til å forlate spillet kort tid etter at det ble sluppet da spillmekanikkene ikke var tilfredsstillende. Dette gjorde at de som hadde bygget spillet ble enige om å ikke utvikle det noe videre, og det ble til slutt skrotet. Drummond et al. (2017) foreslår et rammeverk basert på tidligere forskning som sier at det er fire egenskaper fra kognitiv forskning man bør ha i tankene når man utvikler serious games, disse fire er: Aktiv læring, oppmerksomhet, tilbakemelding og konsolidering. Forfatterne mener at i tråd med serious games betyr aktiv læring at man ønsker å engasjere brukeren med interaksjoner istedenfor å måtte fordøye større mengder tradisjonelt tekstbasert læringsmateriale. Med oppmerksomhet mener de at spillet bør ha en form for veiledning som kan hjelpe brukeren videre dersom vedkommende står fast ved hjelp av symboler, lyder og slikt. Tilbakemelding ønsker å gi brukeren en pekepinn på hvor godt de gjør det, gjerne i form av et poengsystem. Med konsolidering mener de enkelt og greit at det er aktiviteter som kan repeteres. For at læringen skal bli så effektiv som mulig så bør alle disse fire aspektene diskuteres og vurderes under designfasen og planleggingen. Disse poengene sammen med de motiverende effektene må alle være med i regnestykket om man skal oppnå de ønskede læringsmålene.

Frem til nå har fordelene og ulempene med programmering, sluttbrukerverktøy og digitale opplæringsverktøy blitt etablert. Ved å se på hva en domene ekspert er blir også målgruppen og deres krav til språk og grensesnitt enklere å se. Det neste naturlige steget for å kunne besvare forskningsspørsmålet blir derfor å se på forskjellen mellom de to tilnærmingene til visuell programmering. Dette vil dekkes i neste underkapittel.

1.2.6 Hva er forskjellen på blokkbasert og nodebasert programmering?

Selv om både blokkbasert og nodebasert programmering kategoriseres som visuell programmering så er det forskjeller på utformingen og måten disse leses på. For å kunne besvare på forskningsspørsmålet er det vesentlig å hvordan de to tilnærmingene til visuell programmering skilles fra hverandre.

Begge disse formene for visuell programmering skroter den tradisjonelle tekstlige programmeringen til fordel for visuelle elementer. En blokkbasert tilnærming benytter seg av blokker som settes sammen nesten som et puslespill hvor kun enkelte biter passer sammen for å forhindre feil i koden (Weintrop & Wilensky, 2017). Et nodebasert språk vil bestå av noder som gjerne har flere porter for input og output som kobles sammen av et nettverk med tråder hvor informasjonen «flyter» fra node til node gjennom disse trådene (Mason & Dave, 2017). Kode skrevet med en blokkbasert tilnærming vil ofte fremstå som mer oversiktlig og strukturert da det skrives i sammensatte moduler og all kode som henger sammen er samlet på et sted, mens kode skrevet med en nodebasert tilnærming ofte kan se mer rotete og tilfeldig ut da det ikke er noen regler for hvor disse nodene skal plasseres og fokuset heller ligger på hvordan nettverket av noder av konstruert.

Siden man selv velger hvordan nodene kobles kan man se på en nodebasert løsning som mer fleksibel, og det er heller ikke noe i veien for å bruke en output fra en node flere ganger da man kan trekke flere tråder ut av samme node, noe man ikke har muligheten til med kodeblokker som er sveiset sammen. Dette kan gjøre at dersom man jobber med større prosjekter med mer kode hvor det kan være nødvendig med gjenbruk av variabler og funksjonalitet vil koden til et prosjekt laget i et nodebasert grensesnitt kunne være mer kompakt og bestå av færre elementer, riktignok med et mer komplekst nettverk av tråder og koblinger. En hake med konseptet av gjenbruk av kode er at det krever en forståelse av hva koden faktisk gjør i tillegg til evnen til å planlegge og se disse mulighetene, noe som kan være en utfordring for uerfarne programmerere. I tillegg tillater nodebaserte tilnærminger at mer

enn en node prosesserer informasjonen og koden sin samtidig ettersom man kan sende output fra en node til flere andre noder samtidig.

En styrke med enkelte blokkbaserte programmeringsspråk er at det ofte er enkelt å se hvilke deler av koden som passer sammen da disse puslespill-bitene kun kan settes sammen i kombinasjoner hvor bitene passer sammen, som gjør at man minimerer muligheten for å gjøre feil. Dette sikkerhetsnettet er ikke like tydelig i en nodebasert tilnærming, men mange nye verktøy som for eksempel Bolt i Unity lar brukeren se en animert representasjon av hvordan koden flyter gjennom nodene som kan gjøre feilsøking relativt enkelt.

1.2.7 Eksisterende verktøy

Sluttbrukerverktøy er ikke noe som er nytt, og det er gjort forskning på slike verktøy i flere år. For å finne ut hvordan visuell programmering har blitt brukt tidligere er det fornuftig å vende seg til litteraturen for å se hvordan det har blitt brukt i opplæringsverktøy.

I scenario-baserte spill for opplæring og trening er det flere måter å presentere tilpasningsverktøyene til sluttbrukeren. I en artikkel fra 2011 presenterte Marchiori et al. (2011) en løsning for lærere og forelesere med målsetning om å kunne lage scenarioer uten noen forkunnskaper om programmering ved å bruke «state shift diagrams». Dette presenteres til brukeren gjennom et grensesnitt med forskjellige symboler og noder som representerer forskjellige handlinger. Disse nodene er så koblet sammen gjennom et nettverk av linjer og stiplede linjer som representerer overganger mellom handlingene i scenarioet. Et verktøy som dette gir sluttbrukeren en effektiv måte å lage innhold på som kan forgreine seg til en mengde situasjoner innenfor læring samtidig som det gir sluttbrukeren en jevn og fin læringskurve. Da dette er svært enkelt å bruke vil et system uten for komplisert funksjonalitet trolig være best egnet til pek-og-klikk spill eller andre «enklere» former for spill og simuleringer. Forfatterne av artikkelen går ikke inn på hvor godt denne funksjonaliteten dekker behovene man vil ha i et tredimensjonalt miljø eller virtuelle simuleringer. De nevner også Scratch, som er kjent for å være svært brukervennlig verktøy som har blitt brukt til en mengde forskjellige prosjekter fra enkle simuleringer til videospill (Maloney et al., 2010). På samme måte som eksempelet over er også Scratch et grafisk grensesnitt, men er bygget opp av blokker istedenfor noder. Disse blokkene fungerer som biter i et puslespill, hvor det kun er enkelte biter som passer sammen. Dette fjerner muligheten til å produsere kode som ikke vil kjøre på grunn av kodefeil, men vil likevel ikke eliminere feiltolkning i logikken, noe som kan lære brukere

både å finne kreative løsninger samtidig som det er nyttig kunnskap å ta med seg videre i en eventuell overgang til tekstbasert programmering. Scratch har eksistert i nesten tjue år, og har vist å ha en positiv effekt på læringsutbytte for å lære seg programmering.

I 2012 presenterte Marchiori et al. en metode som inkluderer lærere og instruktører i utviklingsprosessen, kalt WEEV (Marchiori et al., 2011). Denne metoden ble benyttet på en allerede eksisterende plattform for spill brukt i opplæring og trening, som ble evaluert og forbedret basert på tilbakemeldinger fra sluttbrukerne. Tanken bak denne metoden er at det finnes tre hovedelementer eller oppgaver som er fordelt i hver sin editor. Disse tre elementene er: objekter, verden og hendelsesforløp. *Objekt* editoren inneholder alle de interaktive elementenesom er brukt i simuleringen, dette kan være alt fra objekter man kan se i rommet man er i eller NPC'er(Ikke-spillerstyrt spiller), som blir importert og lagret i en egen fane i systemet. *Verden* editoren er et verktøy for å tilpasse den virtuelle verdenen som brukeren kan utforske som benytter seg av et DSVL. Sist har vi *hendelsesforløp* editoren som inneholder samhandlingene mellom brukeren og spillet, denne editoren benytter seg også av et DSVL. En av styrkene med en tilnærming som denne er at sluttbrukeren kan kjøre spillet direkte i de forskjellige editorene og se hvordan spilleren beveger seg gjennom grensesnittet før de velger og eksportere ut prosjektet som en kjørbar fil.

1.2.8 Lignende verktøy i andre domener

Som beskrevet i forrige delkapittel blir sluttbrukerverktøy for opplæring brukt en del i skole og utdanning. Forskningen gjort i sammenheng med sluttbrukerverktøy for opplæring i helsesektoren ser stort sett på de overordnede målene med slike verktøy istedenfor de spesifikke teknologiene som blir brukt. Derfor blir det nødvendig å se på lignende verktøy i andre domener og hvordan de har brukt visuell programmering i deres løsninger. Selv om ikke alle de følgene eksemplene omhandler trening og læring så inneholder de eksempler på hvordan blokkbasert og nodebasert programmering har blitt brukt i sluttbrukerverktøy. Det er mange interessante egenskaper som ikke er avhengig av bruk eller arbeidsområde, men som heller kan ses på som generelle punkter når det kommer til sluttbrukerverktøy.

Roboter er ikke lenger science fiction, og blir nå brukt på tvers av flere industrier som assistenter som jobber side om side med mennesker, eller erstatter mennesker helt i enkelte tilfeller. Selv om disse robotene kan være svært effektive, så krever de komplisert programmering for å fungere slik som de skal. Samtidig som disse robotene blir mer og mer

effektive og kostnaden av disse stadig går ned, så går etterspørselen etter slike roboter opp, noe som også fører til at etterspørselen etter personell som kan programmere disse øker. I 2020 ble et studie gjennomført hvor målet var å finne ut om et blokkbasert programmeringsgrensesnitt laget for sluttbrukerne kunne eliminere dette tidligere kravet om å forstå komplisert programmering (Ritschel et al., 2020). Dette ville så kunne lede til at de personene som jobbet side om side med robotene selv kunne programmere oppførselen deres. Forfatterne konkluderte med at dette var svært brukervennlig og engasjerende for deres målgruppe grunnet den enkle blokkbaserte programmeringen. Måten de kom frem til denne konklusjonen på var ved å lage fire forskjellige forslag på verktøy for å se hvilken som resonerte best med de som ikke hadde noen tidligere kunnskap om programmering. Med disse verktøyene skulle de løse tre forskjellige oppgaver med forskjellige vanskelighetsgrader og alle verktøyene skulle brukes til å løse de samme oppgavene. Andre viktige punkter som ble lært av dette prosjektet var at testpersonene foretrakk horisontal programmering over vertikal, og at preferansene til de erfarne testpersonene var slående like de som ikke kunne noe om dette fra før.

Agentsheets var et prosjekt som hadde som mål å senke kravene til forkunnskaper om programmering for å løse programmeringsproblemer. For å oppnå dette ble det brukt en visuell representasjon av et programmeringsgrensesnitt som inneholdt kjente objekter og et domene spesifikt språk med stort fokus på å bruke korrekte abstraksjoner (Repenning & Sumner, 1995). Språket brukte «agenter» som oppfører seg som komponenter som man plasserer i et rutenett. Disse agentene må så gis instrukser om hvordan de skal kommunisere med andre agenter, og dette sammen med plasseringen i rutenettet er to faktorer som sammen definerer syntaksen og semantikken for det visuelle språket. Agentsheets har blitt brukt i flere felt som for eksempel biologi, roboter og informasjonsteknologi for å utvikle interaktive simuleringer på høgskole nivå og trening på arbeidsplassen (Repenning, 2000).

1.3 Sammendrag av kapittel 1

Dette kapittelet har forklart motivasjonen bak prosjektet, definert forskningsspørsmålet og presentert eksisterende teori om sluttbrukerverktøy, utfordringer med å lære programmering, virtuelle simuleringer og serious games i tillegg til en gjennomgang av eksisterende sluttbrukerverktøy som benytter seg av visuell programmering. Litteraturgjennomgangen viser at virtuelle simuleringer og serious games som en læringsplattform viser stort potensiale, og trolig er noe som vil bli mer og mer brukt i fremtiden. En motiverende faktor som kommer

frem i kapittelet om eksisterende verktøy og lignende verktøy i andre domener er at det per i dag virker å være få prosjekter som tar for seg en lignende problemstillings som i dette prosjektet. Eksisterende litteratur virker heller å ha fokus på de overordnede målene med sluttbrukerverktøy for helsesektoren enn hvordan helsepersonell kan håndtere disse verktøyene.

2 Metode

Dette kapitelet vil ta for seg metodene brukt for å kunne besvare forskningsspørsmålet. Kapittelet vil først dekke vitenskapelige metode i overordnede trekk. Dette innebærer hvordan data og hva slags data som samles inn og hvordan dataen behandles etter modellen for kvalitativ forskning fra Creswell og Creswell. Etter denne introduksjonen deles kapitelet opp i fire deler, som alle representerer hver sin fase i prosjektarbeidet. Hver av disse fasene går mer i dybden på utvikling av et nodebasert og blokkbasert programmeringsspråk, hvordan disse blir fremstilt i hvert sitt oppgavesett for å hjelpe oss å samle data og til slutt hvordan selve datainnsamlingen ble gjennomført. Disse fasene vil presenteres i kapittel 2.2.

2.1 Vitenskapelig metode

Da både datainnsamling og analyse vil resultere i hovedsakelig åpne svar og respons fra informantene vil dette klassifiseres som en kvalitativ studie og vil bli gjennomført etter de retningslinjene og arbeidsmetodene dette medfører. En kvalitativ studie har som mål å la forskere studere et konsept eller en teori basert på personers meninger og motivasjoner som kan resultere i innsikt i et problem og en mulig hypotese for hvordan problemet kan løses eller jobbes med videre (Creswell & Creswell, 2018). Sammenlignet med kvantitativ forskning tar kvalitativ forskning ofte for seg en mindre og mer konsentrert gruppe av testpersoner og dataen man analyserer kommer ofte i form av bilder, video og lengre tekstlige svar fra intervjuer eller spørreundersøkelser med åpne svar. Med en problemstilling slik som i dette prosjektet er det mer hensiktsmessig å velge en kvalitativ tilnærming da vi ønsker å få en forståelse for hvordan målgruppen vår oppfører seg og tolker visuell programmering istedenfor en oversikt over riktig eller galt svar.

2.1.2 Informanter og datainnsamling

Datainnsamling i kvalitativ forskning innebærer dokumentasjon blant annet hvor og hvordan innsamling av data skal foregå i tillegg til dokumentasjon under selve brukertestingen (Creswell & Creswell, 2018). Dette underkapittelet vil dekke begge disse delene av prosessen.

Informantene består av studenter og ansatte ved Høgskolen i Østfold sin avdeling for Helse og Velferd i Fredrikstad. Det er noen hovedgrunner til at denne gruppen har blitt valgt. Disse personene er først og fremst potensielle kandidater til å være noen som skal benytte seg av et verktøy som dette i fremtiden, i det minste studentene når de skal ut i jobb etter at utdanningen deres er fullført. Jeg ser det som høyst sannsynlig at dersom enkelte designbeslutninger jeg har tatt underveis er frustrerende for studentene vil det også være det for de som allerede jobber som helsepersonell. En annen karakteristikk ved en kvalitativ studie er at observasjoner tar sted i testpersonenes naturlige miljø istedenfor at de blir hentet inn til en lab eller et ukjent sted (Creswell & Creswell, 2018). Dette er for å se hvordan de oppfører seg i miljøet der aktiviteten man observerer normalt sett vil finne sted, og man kan tillate seg å ha kontakt med testpersonene underveis for å få innsyn i deres tankegang som kan noteres å tas med videre til analyse.

Det er også normalt at man har flere kilder for datainnsamlingen, altså at man samler data ved å bruke flere metoder. Datainnsamlingen deles derfor inn i to deler: En digital innsamling hvor studenter og ansatte ved høgskolen sin avdeling for Helse og Velferd får delta og en fysisk innsamling hvor ansatte hovedsakelig tilknyttet skolens simuleringssenter deltar. Da den digitale innsamlingen måtte være anonym i tråd med den godkjente søknaden som ble sendt inn til Norsk senter for forskningsdata(NSD) måtte jeg benytte meg av gruppesider på sosiale medier og rekruttering gjennom nøytrale tredjeparter for å nå ut til informanter. Denne prosessen viste seg å ta lenger tid enn opprinnelig planlagt da det var utfordrende å rekruttere informanter. Heldigvis betyr også dette at de som valgte å gjennomføre oppgavene var genuint interessert i å bidra til fremdrift i forskningen min, og dermed resulterte i stort sett fulle besvarelser. Som tidligere nevnt er et trekk ved kvalitativ forskning at observasjonene finner sted i testpersonenes naturlige miljø, derfor fant den fysiske datainnsamlingen sted på høgskolen sine lokaler i Fredrikstad. Ønskelige kandidater ble kontaktet over e-post, og de som aksepterte ble invitert til et fysisk møte. Ved å ha begge disse formene for datainnsamling kommer dataen fra flere forskjellige kilder som kan tas med videre til analyse.

2.1.3 Analyse av data

Poenget med å analysere dataen er å skape et bilde og en mening med all informasjon som nå har blitt samlet inn. Creswell og Creswell presenterer en modell som inneholder fem steg de mener man bør følge når man skal analysere kvalitativ data (Creswell & Creswell, 2018).

1. Det første steget innebærer å sortere og klargjøre data for analyse. I denne prosessen skal man lagre alt bildemateriale på et sted, transkribere intervjuer og organisere data etter kategori og datatype. Dette steget er kritisk for å spare tid til senere da biblioteket av data man lager benyttes i resten av analyseprosessen.
2. Følgende steg består av å gå gjennom alt av innsamlet data for å få en generell forståelse av resultatet av datainnsamlingen i form av tanker og følelser fra informantene. I dette steget er det også normalt at man begynner å loggføre sine egne tanker og ideer om resultatene man nå sitter med i form av notater i dokumenter, lydopptak av seg selv eller andre metoder man måtte foretrekke. Man bør om mulig danne seg et oversiktlig bilde over kredibiliteten og omfanget av den innsamlede dataen.
3. I steg tre skal man kode all dataen sin. Denne prosessen innebærer å hente ut deler av tekst og bilder av dokumentene for så å kategorisere disse med merkelapper med såkalte «in vivo» nøkkelord, som baserer seg på ord og utrykk som går igjen hos testpersonene. I motsetning til sorteringen av data i første steg skal man her gå dypere inn i hvert enkelt bilde, dokument og slikt for å ytterligere kategorisere og spisse dataen din.
4. Etter å ha kodet dataen skal denne brukes til å lage beskrivelser om ting som steder, personer og hendelsesforløpet i datainnsamlingen. Beskrivelsene bør ta flere personers perspektiv og sitater i betraktning som kan settes opp mot hverandre og diskuteres. Målet med dette steget er å sitte igjen med en håndfull temaer som kan hjelpe med å besvare forskningsspørsmålet.
5. Sist skal man sette sammen alle de foregående stegene til å presentere resultatene i et narrativ. Dette narrativet kan for eksempel presenteres som en diskusjon av resultatene i kronologisk rekkefølge, en diskusjon om noen av de viktige temaene man har jobbet seg frem til og lignende. I dette steget kan man også dele inn resultatene i tre kategorier:
    a. Forventede funn: Denne kategorien inneholder de tingene man kanskje så komme eller som man kunne tenkt seg frem til med sunn fornuft.

b. Overraskende funn: Som navnet på kategorien tilsier representerer denne kategorien de funnene man ikke på forhånd kunne ha foresett. Dette er resultater man ikke kunne ha tenkt seg frem til uten å gjennomføre omfattende forskning.
c. Uvanlige og konseptuelle funn: Noe lignende den foregående kategorien, men er mer rettet på resultater og konsepter som kanskje er av interesse for leseren men som ikke nødvendigvis er viktige for å besvare forskningsspørsmålet.

Disse kategoriene blir presentert i kapittel 3. Ved å følge denne modellen vil analysen bli strukturert basert på en metode som er godt dokumentert og anbefalt av meriterte forskere. Dette vil legge grunnlaget for en sømløs innsamling av data, i tillegg til klare retningslinjer for hvert steg i analysen.

2.2 Fase 1: Innsikt – Gjennomgang av verktøy

I litteraturgjennomgangen dekkes flere verktøy som bruker visuell programmering for opplæring eller støtte i arbeidsoppgaver. Selv om kapittel 1.2.7 og 1.2.8 tar for seg sluttbrukerverktøy som benytter seg av visuell programmering er det ikke gitt at løsningene de har gått for i de prosjektene er de som er best for å kunne manipulere scenariobasert innhold. I dag er disse modifiseringsverktøyene trolig noe av det nærmeste man kommer noe som tilbyr funksjonalitet som kan sammenlignes med scenariobaserte virtuelle simuleringer. Modifisering av dataspill kan nesten regnes som en sjanger innenfor spillverdenen i seg selv, og de verktøyene som blir brukt for å lage, endre eller forbedre innhold benytter seg av forskjellige tilnærminger basert på hvor komplekse operasjoner sluttbrukerne skal gjøre. Det som er interessant med disse verktøyene er at de er tilgjengelig for alle. Hvem som helst som eier spillene kan laste ned eller få tilgang til disse verktøyene, og de som lager disse verktøyene vet aldri hvem som sitter på andre siden og tar det i bruk.

De verktøyene som har blitt testet og analysert er håndplukket etter å ha ført en diskusjon over e-post om denne typer verktøy med Jarl Schjerverud, tidligere ansatt hos Funcom og på Sykehuset Østfold som prosjektleder i spillteknologi. En ting som er verdt å nevne før de forskjellige verktøyene blir analysert er at det finnes et flertall av blokkbaserte verktøy for modifisering av spill i forhold til nodebaserte verktøy. De nodebaserte grensesnittene virker å være mer brukt som individuelle deler av verktøy som håndterer deler av en programvare

istedenfor at alt er bygget rundt disse nodene. Gjennomgangen av disse verktøyene vil være med på å legge grunnlaget for fase 2 i prosjektet, hvor første iterasjon av programmeringsgrensesnittet blir laget.

2.2.1 Creation Kit

Creation Kit er en relativt kompleks programvare laget av Bethesda som lar spillere modifisere spill som The Elder Scrolls V: Skyrim eller Fallout 4. Dette verktøyet blir også brukt inhouse hos Bethesda, noe som gjør at hvem som helst potensielt kan lage modifikasjoner med kvalitet på linje med utviklerne selv. Scripting i Creation Kit gjøres med hjelp av deres egne script språk, kalt Papyrus. Det som fanget interessen min mest med Creation Kit var dialog-verktøyet deres. Ved å bruke dette verktøyet kan spillerne selv lage dialog med NPC'er, og til og med spille inn egen dialog som blir animert automatisk, slik at munnen til karakterene du snakker med beveger seg samtidig som de sier noe. Her møter man på første svakhet med dette verktøyet; nemlig at man trenger flere programmer for å gjøre enkelte ting. Når man for eksempel tar opp lyden til en dialog, vil dette blir lagret i separate filer, mens filene for animasjonen av munnen deres blir lagret samme sted i filstrukturen, dog også som en separat fil. Når man skal implementere denne dialogen i selve spillet må disse filene slås sammen til en såkalt .fuz fil, noe som ikke gjøres i Creation Kit, men derimot må gjøres med et eget program man må ha installert som heter UnFuzzer. Selv om dette i seg selv ikke nødvendigvis er problematisk for erfarne utviklere og de som kjenner til denne typen verktøy, så forstår jeg at dette kan anses som en kompleks løsning for nybegynnere, og dette burde etter min mening blitt gjort av seg selv uten hjelp av eksterne verktøy. Et annet aspekt med Creation Kit som jeg kan se for meg kan være noe problematisk for de som er nye i bruken av dette verktøyet er at det opererer med mange vinduer, dette kom veldig tydelig frem i dialog verktøyet som jeg utforsket mest. Du har et nettverk satt sammen av noder med dialogvalgene du lager som ligger i bunnen, mens for å få tilgang til enkelte funksjoner, som for eksempel det tidligere nevnte verktøyet for lydopptak, vil dette komme som et eget vindu. Etterhvert som dialognettverket blir større vil også mengden av disse vinduene bli fler og fler etterhvert som du trenger funksjoner fra hver enkelt. Ikke bare ser det skremmende ut for nye brukere, men kan også virke forvirrende selv for de med erfaring. En av de bedre funksjonene Creation Kit tilbyr er at man kan legge inn flere responser på samme dialog, altså at du ikke vil få samme svar hver gang du velger å føre den samme dialogen. Dette gir scenene mer liv etterhvert som man går gjennom de flere ganger, og er enkelt gjort ved at man gjør flere lydinnspillinger og velger respons til å være «random»

2.2.2 Game Master Mode

I spillet Divinity: Original Sin II har utviklerne inkludert en egen Game Master Mode i grunnpakken av spillet, heretter kalt GMM. GMM er langt mindre kompleks enn Creation Kit og vil for mange, spesielt de som ikke har noe erfaring med programmering og utvikling fra før, være enklere å lære seg sammenlignet med mange av de andre alternativene jeg har sett på. En av de sterkeste sidene jeg ser med dette verktøyet er at man aldri blir tatt ut nivået man lager og sendt inn i en annen editor. På denne måten ser man til enhver tid hvordan endringene man gjør påvirker scenen, da alt av karakterer og gjenstander man plasserer umiddelbart havner på skjermen. Målet med nivåene man lager er at man enkelt skal kunne dele disse med sine venner over nett, hvor man får muligheten til å spille gjennom nivåene mens en spiller har rollen som «game master». Som «game master» vil man fortsatt ha mange av verktøyene tilgjengelig slik at man kan gjøre endringer i scenen selv mens man spiller, og jeg mener dette sier sitt om hvor brukervennlig dette verktøyet er. På samme måte som Creation Kit opererer GMM med flere vinduer for forskjellige operasjoner. Man har for eksempel et eget vindu åpent for å endre karakteristikken til NPC'er man møter, mens man har en editor for å lage beskrivelser og verdier til gjenstander i et annet. Forskjellen her er at alle vinduer har en dedikert plass på skjermen i tillegg til at de er bygget inn selve arbeidsområdet istedenfor at det åpnes et helt eget vindu. Dette gjør at man til enhver tid har oversikt over hvilke vinduer som er åpne - dersom det ikke tar opp plass på skjermen din akkurat nå, så er det ikke i bruk. Ikke bare er dette tidsbesparende, men også betryggende på den måten at man ikke blir overveldet av informasjonen og knapper man kan trykke på. I GMM er det ingen form for programmering. Alt av elementer og verktøy er istedenfor tilgjengelig i menyer som plasseres på et arbeidsområde ved hjelp av drag-and-drop, som leder til at verktøyet kan brukes av så og si hvem som helst uten noen forkunnskaper. En annen fordel er at det finnes forhåndsdefinerte karakterer i form av fiender, handelsmenn og mange fler som man kan plassere i scenen sin for så å enkelt tilpasse for eksempel hva slags varer en handelsmann har, eller hva slags belønning man får av å drepe et monster.

Selv om alt frem til dette punktet høres ut som positive egenskaper har naturligvis den simple naturen til GMM gått på bekostning av ting som funskjonalitet og fleksibilitet. For å nok en gang trekke sammenligninger til det foregående eksempelet med Creation Kit er det ikke noen form for dialogverktøy i det hele tatt. Man kan lage beskrivelser på gjenstander og få tekstlige beskrivelser underveis, men man kan ikke føre en dialog med svaralternativer og konsekvenser på lik linje med Creation Kit. Jeg ser ikke på dette som en mangel for selve

opplevelsen i spillet da narrative skal drives av en «game master» som forteller historien underveis. Alt man kan lage som karakterer og gjenstander er også basert på ressurser som er levert med spillet. Man kan med andre ord lage egne beskrivelser og justere verdier på gjenstander, men visuelt sett er man begrenset til modeller og ikoner som følger med i grunnspillet. Jeg har ikke spilt nok av disse scenarioene selv, men kan se for meg at denne faktoren kan gjøre at det kan bli noe repetitivt dersom man spiller gjennom flere ganger. Den samme svakheten møter vi med animasjoner.

2.2.3 Mcreator

Minecraft er en gigant i «modifiserings-verdenen» hvor det har blitt laget utallige modifikasjoner av spillere verden over som gjør alt fra å gi spillere nye blokker og nytt design på blokker til grafiske overhalinger av hele spillet. Mcreator er et av flere verktøy laget for å lage modifiseringer til spillet, og sammenlignet med de to forrige eksemplene er ikke dette verktøyet laget av utviklerne av spillet, men heller av en tredjepart. Sett fra et estetisk ståsted står Mcreator nærmest det vi vil kjenne igjen fra de populære tekst-editorene som blir brukt til programmering. Mcreator inneholder likevel et verktøy for blokkbasert programmering som kalles Procedure, som tar bort flere av kravene om å kunne programmere. Selv om Procedure benytter seg av ting som logiske operatorer og løkker så er de forenklet til et nivå hvor de som er godt kjent med Minecraft trolig vil forstå seg på mye av det verktøyet tilbyr. Dette har utviklerne fått til ved at for eksempel variabelnavn kan være endret til små bilder av blokkene som er i spillet eller at parametere er endret til navn på objekter eller operasjoner spillere vil kjenne seg igjen i. Mens dette er en fordel for de som ikke kan programmere fra før kan jeg se for meg at dette kan være noe forvirrende for de som enten er erfarne innenfor feltet og er vant med for eksempel «vanlige» variabelnavn eller for sånne som meg som ikke er noe kjent med Minecraft fra tidligere hvor disse symbolene ikke forteller brukere utenfor målgruppen stort. Flere av disse blokkene er også ferdig laget, slik at mange av operasjonene ikke trenger å bli laget manuelt, men man har likevel muligheten til å endre på verdier slik at de gjør akkurat det som ønskes.

På lik linje med Scratch er også blokkene satt sammen som puslebiter, noe som gjør det enklere å se hva slags blokker man kan bruke, hvor man kan bruke dem og samtidig eliminere muligheten til å gjøre noen feil i mange tilfeller. Dette gjelder ikke nødvendigvis feil i logikken, men heller kodefeil man kan oppleve i tradisjonell tekstbasert programmering. Samtidig som man har muligheten til å kode i Procedure tillater også Mcreator at man skriver

kode i Java for de som skulle ønske det. En annen god egenskap med Mcreator er at før man i det hele tatt starter å lage noe så kan man si til programmet hva du skal lage, så vil automatisk de funksjonene man ikke skal bruke bli fjernet fra verktøykassen din. Dette betyr at om man for eksempel skal lage en ny type blokk vil man kun få tilgang til de verktøyene som er relevant for dette problemet. Dersom man skal finne et verktøy å hente inspirasjon fra for blokkbasert programmering med en noe tilsvarende hensikt og målgruppe vil jeg se på som Mcreator som et godt sted å starte.

2.2.4 Unreal engine – Blueprint scripting

Som nevnt innledningsvis var det noe utfordrende å finne nodebaserte verktøy rettet mot samme mål- og produktgruppe. Derfor har jeg valgt å hente et eksempel utenfra spill modifiserings-verdenen, da i form av Unreal Engine sitt såkalte «Blueprint visual scripting» system. Ulempen med denne sammenligningen er at dette ikke er siktet på sluttbrukere på samme måte som de tidligere eksemplene da det ikke er rettet mot et spesifikt spill og personer som interesserer seg for dette, men heller er en spillmotor man kan lage spill i. I tillegg er det også en mer omfattende programvare å sette seg inn i, og mulighetene for hva man kan lage er betydelig større. Dette scripting systemet ble laget for å gi grafiske designere og andre uten omfattende programmeringskunnskaper mulighet til å programmere selv, i dag er systemet godt nok utviklet til at man kan lage hele, dog enkle spill uten å skrive en eneste linje med tekstbasert kode (Chu & Zaman, 2021). En styrke med blueprints er at etterhvert som man lager fler og fler blueprints kan man bruke disse om igjen, så dersom man ønsker å ha flere av samme ting i et level, for eksempel en dør, så trenger man bare å lage en blueprint til dette en gang for så å gjenbruke det. Man lager seg med andre ord et bibliotek av assets, både i form av kode og objekter. Blueprints er som nevnt tidligere satt sammen av et nettverk av noder, hvor nodene er fornuftig kategorisert med fargekoder avhengig av hva de gjør. På samme måte som med tradisjonell programmering har man også mulighet til å kommentere «koden» sin, noe som vil hjelpe erfarne programmere med å kunne ta over prosjekter andre har jobbet på tidligere, samtidig som det er en god vane å legge til seg for de som ikke kan så mye fra før. En annen styrke med blueprints er at etterhvert som nettverket av noder blir større og mindre oversiktlig, så er det en søkefunksjon inkludert i Unreal Engine, som enkelt lar brukerne navigere og lokalisere tidligere arbeid. Denne funksjonen ser selv på blueprints man ikke har åpne, noe som gjør at man kan få en ryddig arbeidsflyt. Sammenlignet med de foregående eksemplene er dessverre blueprint systemet milevis foran de andre i kompleksitet, noe som kan være avskrekkende for nye brukere. Syntaksen her er mer lik det man kjenner

igjen fra tradisjonell programmering, og sammenlignet med for eksempel Mcreator hvor mange av blokkene inneholder informasjon fra Minecraft, er det opp til brukeren selv å lage og definere blueprints. Dette kan føre til at de med lite eller ingen erfaring med programmering må bruke mer tid på å lese dokumentasjon og potensielt miste interessen da man ikke ser resultatene av det man gjøre like raskt.

2.2.5 Unity - Bolt

Bolt er en utvidelse til spillmotoren Unity som lar brukerne programmere alt fra hvordan en karakter oppfører seg til hendelsesforløp. Bolt ble nylig integrert som en del av grunnpakken du får når du laster ned Unity, noe som ikke var tilfellet med versjoner eldre enn 2021. Dette er på samme måte som Blueprint scripting et nodebasert programmeringsgrensesnitt, men opererer likevel på en annen måte ved at det er mer likt programmeringsspråket spillmotoren er basert på. Bolt er med andre ord en visuell representasjon av C#. Dette alene gjør læringskurven for Bolt noe høyere enn de andre verktøyene vi har sett på. Sett bort fra den høyere kunnskapsbarrieren som kreves for å effektivt ta i bruk Bolt er det flere gode egenskaper ved dette verktøyet som kan hjelpe sluttbrukerne å skrive kode. Den beste kvaliteten med Bolt i mine øyne er måten koden din blir grafisk representert mens den kjøres. Så fort du setter Unity i «play mode» kan du se i programmeringsgrensesnittet ditt hvordan informasjonen flyter fra en node til en annen. Denne flyten blir visualisert med animasjoner, og det er representert enkelt og greit med sirkler som følger trådene fra node til node. I tillegg til dette lar Bolt brukerne sine endre koden mens den kjøres. Dette gir brukerne mulighet til å veldig enkelt se konsekvensene av koden de «skriver», samtidig som det lar brukerne enkelt gjøre små justeringer uten å hele tiden måtte starte og stoppe programmet. Koden man lager med Bolt kan med litt arbeid fremstå som ryddig og oversiktlig, dette er takket være muligheten man har til å markere og navngi større områder med kode. Disse større samlingene med kode kan også enkelt kopieres og gjenbrukes flere ganger i samme prosjekt.

2.2.6 Sammendrag av fase 1

I den første fasen av prosjektarbeidet ble det gjort en gjennomgang av modifiseringsverktøy for dataspill for å lage en grunnleggende forståelse av hvordan eksisterende verktøy lar sluttbrukere manipulere hendelsesforløpet i spill. Fem verktøy ble testet: Creation Kit, Game Master Mode, Mcreator, Blueprint Scripting i Unreal Engine og Unity sin tilleggspakke Bolt. Alle disse verktøyene har styrker og svakheter som blir tatt med videre til fase 2. Da alle verktøyene deler egenskapen med at det er lett tilgjengelig for alle virker det som at det er en

fellesnevner at flere av verktøyene har en form for et domene spesifikt språk rettet mot spillet de hører til. Dette gjelder i mindre grad de nodebaserte verktøyene, men til gjengjeld tilbyr de et lesbart og fleksibelt programmeringsgrensesnitt.

Det virker i stor grad som kompleksiteten og omfanget av verktøyene regulerer graden spillene kan manipuleres i. Eksempelvis kan verktøy som Bolt og Blueprint Scripting la brukeren bygge opp hele spill fra bunnen av, mens GMM i stor grad er begrenset av materiale verktøyet kommer med. Tre hovedpoeng som blir tatt med videre fra denne fasen er: ryddig grensesnitt, kategoriserte og/eller fargekodede kodeelementer og viktigheten av fornuftig semiotikk.

2.3 Fase 2 – Idéutvikling: Første iterasjon av programmeringsspråk

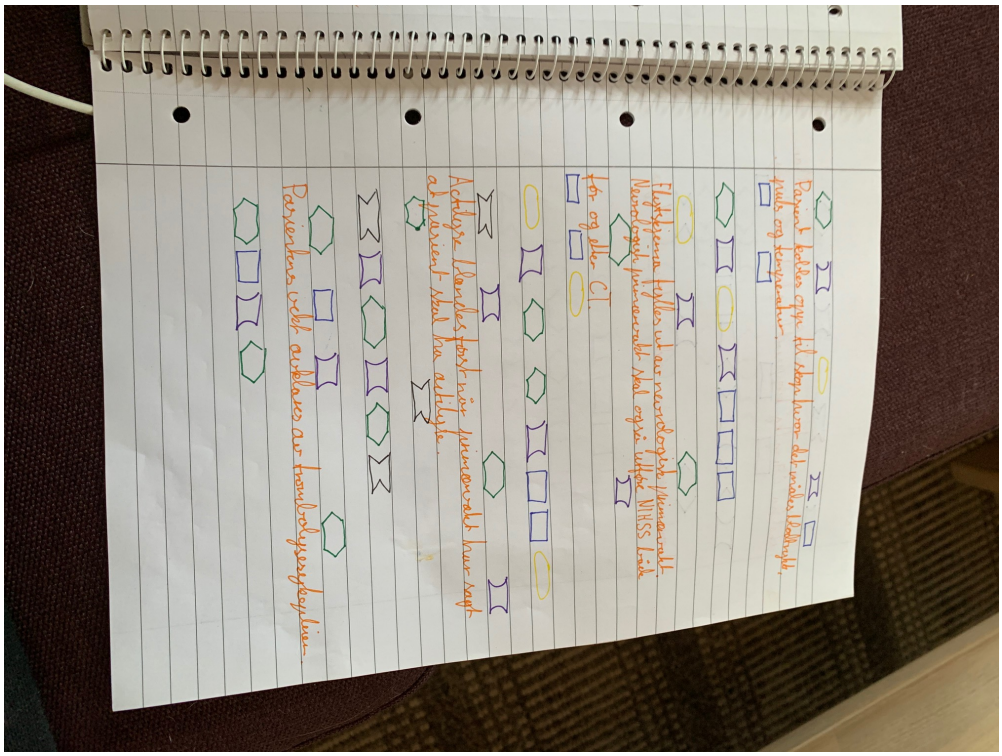
Fase 1 ga en grunnleggende forståelse for hva de forskjellige verktøyene er kapable til å gjøre og skaper et bilde på hvordan kompleksiteten i verktøyene regulerer funksjonaliteten. Etter å ha sett på flere verktøy i litteratursøket og i fase 1 av prosjektarbeidet startet idéutviklingen til det blokkbaserte programmeringsspråket. Istedenfor å lage et språk fra bunnen av med de komponentene og konseptene man vanligvis møter i tradisjonell programmering ble det tatt utgangspunkt i et medisinsk scenario. Det ble også 3D-printet en prototype av dette programmeringsspråket med et grensesnitt som skulle blitt brukt til en fysisk datainnsamling. I dette delkapittelet finnes en detaljert beskrivelse av utviklingen av første iterasjon av språket.

2.3.1 Blokker og språk

For å lage et språk informantene kan programmere i var det en naturlig beslutning å først se på hva slags problemer som skal løses, og hvilke elementer som kreves for å løse disse problemene. Fra et tidligere masterprosjekt ved høgskolen ble det gjort en evaluering av simuleringer for et verktøy brukt for opplæring for håndtering av trombolyse sløyfe. Samme prosedyre ble valgt for dette prosjektet da prosedyren er godt dokumentert i denne oppgaven. Første steg er å oversette prosedyren til et språk som også kan forstås av personer som ikke er eksperter i feltet da prosedyren inneholder mye faglig terminologi. Gjennom denne prosessen ble prosedyren delt opp i flere steg, for eksempel «forberedelser», som tar for seg tidsrommet før pasienten ankommer sykehuset. Hvert av disse stegene har en punktliste med de

forskjellige oppgavene som skal gjøres i løpet av denne delen av prosedyren, som alle er beskrevet så enkelt som mulig.

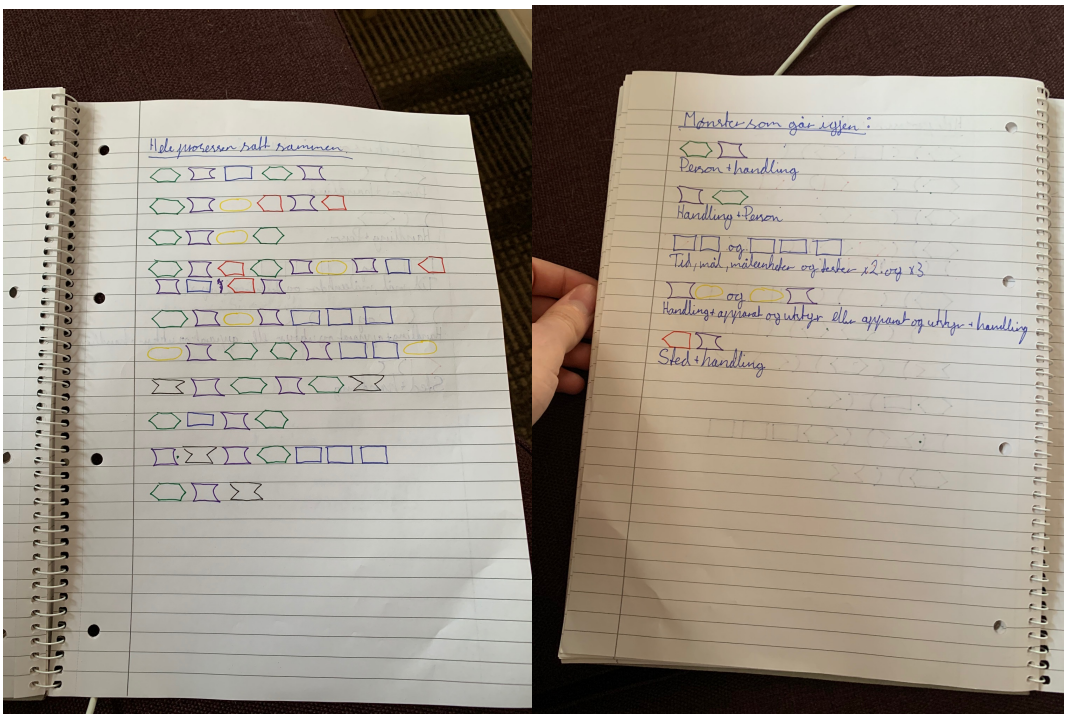
Teksten ble analysert og brutt ned ord for ord, hvor hvert ord ble plassert i en kategori med tilhørende fargekode. Disse kategoriene inkluderer elementer som personer, legemidler, apparater og utstyr.



Figur 2.1 Scenario brutt ned i forskjellige kategorier

Etter å ha kategorisert alle elementene av prosedyren ble prosedyren gjenskapt ved å kun bruke de blokkene som var tilgjengelige. Det var flere grunner til å gjøre dette, men målet var først og fremst å se om blokkene som på dette tidspunktet hadde blitt laget var tilstrekkelig for å kunne gjenskape prosedyren, for så å lage nye blokker der det var mangler. For å gjøre språket noe mer naturtro ble det også produsert flere blokker som ikke hører hjemme i akkurat denne prosedyren, men som er naturlig å ha med i et slikt verktøy. Noe som kom tydelig frem under denne testingen var at det var tydelige mønstre som gikk igjen. Man så ofte at de samme blokkene havnet etter hverandre, eksempelvis dukket ofte en lilla og en grønn blokk etter hverandre, som representerer en handling og en person. Mønstre som dette er naturlige når man tenker over det i etterkant, men det dukket opp flere av disse som man ikke nødvendigvis hadde sett eller ofret en tanke med mindre man ser det foran seg.

Blokkene i første iterasjon har ikke bare forskjellig farge, men også unike former. Disse kan i enkelte tilfeller kobles sammen, men dette er ikke hensikten med formene. En artikkel av Morrison et al. (Morrison et al., 2020) som tar for seg utfordringene med læring av svaksynte barn og læring av programmering i skolen fikk meg til å tenke på eventuelle fallgruver med designet. Selv om informantene i dette prosjektet trolig ikke har fullt så omfattende vansker med synet slik som de som nevnes i artikkelen er det for eksempel ikke unormalt at personer er fargeblinde. For ikke å støte på problemer hvor brukertestingen blir påvirket av hvorvidt testpersonene er fargeblinde eller ikke har de derfor forskjellige former slik at de enkelt kan skilles fra hverandre. Dette er samme løsning som Morrison et al. brukte i sitt prosjekt, Torino, som benyttet seg av kuleformede objekter hvor noen var eksempelvis mer avlange eller runde enn andre, hvor hver enkelt type form hadde en unik egenskap.



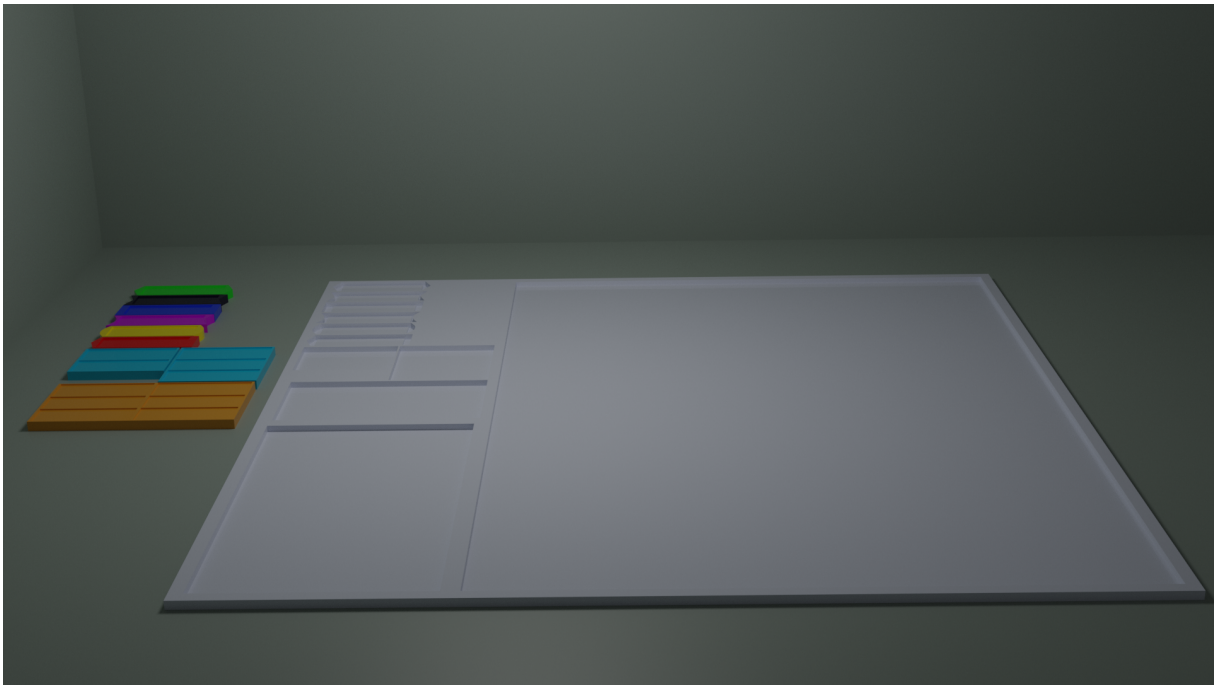
Figur 2.2 Scenario gjenskapt med kategoriene fra blokkene.

Tatt de tidligere nevnte mønstrene i betraktning ble det gjort en avgjørelse om at det skulle bli laget tre varianter av språket med forskjellige grader av kompleksitet. Den første varianten er så enkel som mulig, med svært lite frihet og ferdig sammensatte blokker. I denne varianten var de fleste stegene allerede er ferdig satt sammen av en de nødvendige kombinasjonene man trenger, og brukeren trenger kun å fylle inn de ordene som mangler. Den andre varianten tilbyr mer frihet kan inneholde ferdig sammensatte par slik som eksemplene over med en grønn og lilla blokk men at brukeren likevel må ta mer kontroll å sette sammen funksjonene

selv. Til slutt har vi en variant som tilbyr total frihet, hvor den eneste begrensningen man har er de brikkene man har fått tildelt og det er opp til hver enkelt å konstruere alle funksjoner selv. De tre prototypene kan alle brukes til å løse det samme problemet, men krever hver sin tilnærming og tankegang fra brukerne.

2.3.2 Første prototype

Tidlig i prosjektet var planen å ha en ekstra datainnsamling med den første iterasjonen av programmeringsspråket. Det ble derfor 3D-modellert et programmeringsgrensesnitt med blokkene som på dette tidspunktet var laget. Uheldigvis ble aldri dette ferdigprodusert eller testet grunnet komplikasjoner i sammenheng med covid-19 pandemien. Dette underkapittelet vil kort dekke denne delen i prosessen.

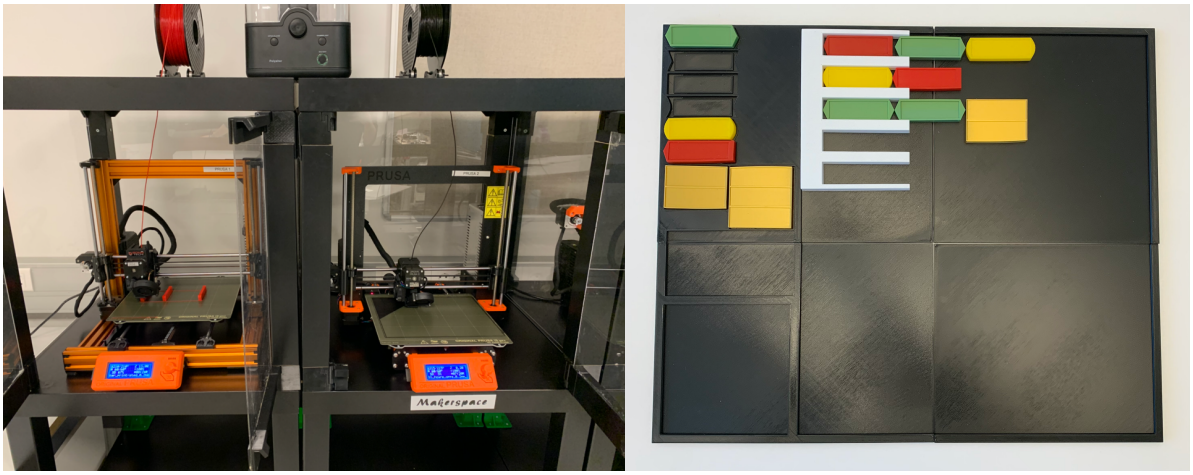


Figur 2.3 3D illustrasjon av fysisk prototype, laget i Blender

Første del av modelleringen gikk med på å gjøre om de flate 2d-skissene til 3d-modeller, noe som i grunn gikk enkelt og greit, med unntak av noen ting som ikke hadde blitt forutsett. Den største utfordringen kom i form av muligheten til å lage menyer og undermenyer for de forskjellige blokkene som falt bort grunnet fysiske begrensninger man ikke har i et digitalt miljø. Flere løsninger for dette ble testet, men jeg landet til slutt på å erstatte området der det ellers ville vært en meny med blokker til å ha former for hver enkelt type blokk slik at testpersonene hele tiden vil se hva som er tilgjengelig for dem. For å kunne fylle inn verdiene i de forskjellige blokkene ble det produsert små biter av stiv papp som er fargekodet og

inneholder alle variabelverdiene tilhørende de forskjellige kategoriene. Disse bitene av papp skal legges i et eget felt i hver enkelt brikke som er formet som et rektangel inne i selve brikken.

Produksjonen av denne prototypen ble aldri ferdigstilt da skolen stengte ned to dager etter at denne prosessen ble startet. Nedenfor finnes to bilder av hva som ble produsert før ideen til slutt ble lagt på is da utstyret ikke lenger var tilgjengelig. Dette ledet til at datainnsamlingen i sin helhet måtte baseres på digitale oppgavesett, som vil beskrives i fase 3.



Figur 2.4 3D printing og 3D-printede elementer

2.3.3 Sammendrag av fase 2

I fase 2 ble første iterasjon av programmeringsspråket produsert med inspirasjon fra verktøyene som ble utforsket i fase 1. Konstruksjonen av språket tok utgangspunkt i en medisinsk prosedyre, nærmere bestemt trombolyse sløyfe, hvor målet med denne fasen var å lage et programmeringsspråk som kunne gjenskape denne prosedyren. Det ble 3d-printet et grensesnitt og flere blokker som skulle vært en del av en innledende brukertest, men denne måtte bli lagt på is grunnet restriksjoner i forhold til pandemien. Denne prototypen håndterer kun naturlig språk og vil ikke være tilstrekkelig til å dekke hele datainnsamlingen i prosjektet, men det var likevel mye å lære fra denne fasen. Det første og viktigste punktet som tas med videre handler om mønstrene som stadig dukker opp i koden. Disse gir en indikasjon på nøkkelord som ofte havner sammen, noe som tas med videre til fase 3.

2.4 Fase 3 – Utvikling: Digitalt oppgavesett og første datainnsamling

Prototypen som ble halvveis utviklet i forrige fase hadde en hake som gjorde at den alene aldri kunne stått for hele datainnsamlingen til prosjektet. Denne haken var at programmeringsspråket på dette stadiet i prosessen kun opererte med naturlig språk. For å kunne manipulere scenariobaserte virtuelle simuleringer er det også vesentlig å kunne kommunisere med datamaskinen. Denne funksjonaliteten som ble introdusert i de digitale oppgavesettene vil presenteres i denne fasen. En oversikt over den innsamlede dataen fra den digitale innsamlingen er også dokumentert i denne fasen.

2.4.1 Digitalt oppgavesett

At fysiske møter ikke lenger var gjennomførbart gjorde at datainnsamlingen måtte la informantene svare digitalt. Da restriksjonene senere lettet ble det også gjennomført en fysisk datainnsamling som er dokumentert i fase 4. I sammenheng med restriksjonene ble derfor to digitale oppgavesett utviklet som tok for seg henholdsvis nodebasert og blokkbasert programmering. Fase 2 i prosjektet la grunnlaget for hvordan disse språkene ble utformet og presentert i oppgavesettet. Som nevnt i kapittel 2.3.1 var det flere mønster som viste seg å gå igjen med blokker som ofte ble brukt sammen. Dette funnet var det viktigste som ble tatt med inn i konstruksjonen av oppgavesettene hvor målet var å få skrevet så mye kode som mulig, med så få komponenter som mulig. I tillegg til dette ble fargekodene fra første iterasjon tatt med videre inn i oppgavesettet. Blokkene og nodene er i stor grad likt utformet, og forskjellen på dem ligger hovedsakelig i måten de kobles sammen og leses på. Måten dette er løst på er først og fremst basert på det som virker å være godt etablert i litteraturen og de verktøyene som ble utforsket i fase 1.

At det gikk fra en fysisk til en digital løsning vil si at noen ting må gjøres annerledes og fokuset på å øke informantenes forståelse må vektlegges da de ikke får muligheten til å stille spørsmål som skulle dukke opp underveis i den digitale datainnsamlingen. Besvarelsene på oppgavene vil også være anonyme, noe som kan ses på som både en fordel og en ulempe. På den positive siden vil ikke veiledning ha noen påvirkning på besvarelsene. Noen ville naturlig nok ville ha stilt flere spørsmål enn andre under et intervju, noe som kunne ha gjort at noen hadde fått fordeler over andre. I tillegg til dette vil det også være positivt at alle besvarelser blir lagret i sin helhet, sammenlignet med den fysiske innsamlingen hvor intervjuet må dokumenteres i etterkant. Noe negativt med den digitale datainnsamlingen er at informantene

kan svare det de selv ønsker og det er ingen selvfølge at de svarer på det oppgavene spør om dersom de skulle misforstå noe. I de fysiske intervjuene er det mulig å fiske etter den informasjonen som ønskes dersom det viser seg at fokuset hos enkelte informanter ligger på andre områder enn det oppgavene ønsker å utforske. Når det er sagt så kan dette også gi muligheten til å få innsikt som ellers ville blitt oversett eller lagt til side.

Oppgavesettene er laget i Google Skjemaer. Denne plattformen tillater utforming av både oppgaver og svar i mange formater og fremstår som oversiktlig og enkel. Med tanke på at hvem som gjennomfører oppgavene er en ukjent faktor bortsett fra hvilken målgruppe de er i er materialet presentert så enkelt som mulig å for å unngå at de som ikke er særlig datakyndige skal støte på utfordringer som demotiverer eller skaper irritasjonsmomenter som vil gå utover besvarelsene. Utover dette så tillater plattformen at data samles og presenteres på en fornuftig måte som gjør det enkelt å ta det med seg videre til analyse. Google Skjemaer tillater forøvrig også besvarelser med opplastning av filer, noe som benyttes i siste oppgave av settet. Oppbygningen og utformingen av oppgavene vil bli beskrevet i detalj i følgende underkapittel.

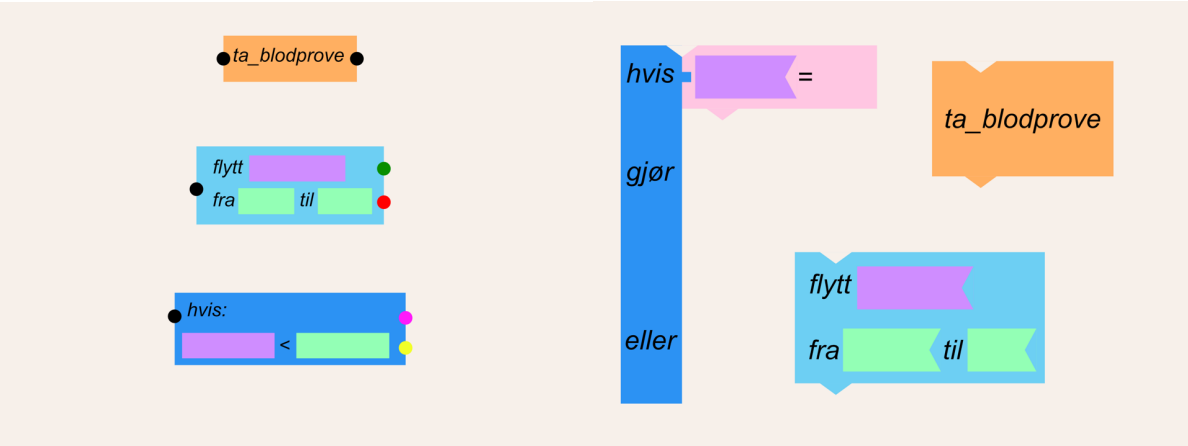
2.4.2 Oppbygning av oppgavesett

Oppgavesettene består totalt av 18 sider hvorav 6 av disse er oppgaver. De fleste informasjonssidene består av både tekst og video som inneholder samme materiale, slik at informantene selv kan velge hvordan de skal få informasjonen presentert.

De første sidene inneholder kun informasjon. Det hele starter med en forenklet beskrivelse av prosjektet og formålet med oppgavesettet. Oppgavesettet leder så videre til informasjon om hva et sluttbrukerverktøy er og motivasjonen bak prosjektet. Resten av oppgavesettet inneholder informasjon om hva programmering er og hvordan det fungerer i tillegg til instruksjoner om hvordan blokkbasert og nodebasert programmering fungerer. Det starter med enkle konsepter om hvordan man representerer naturlig språk til brukeren i form av knapper og menyvalg. Datamaskinen trenger naturligvis også instruksjoner, og informantene får informasjon om både hvordan man kan gi en maskin de instruksene den trenger i tillegg til hvordan man kobler naturlig språk sammen med disse instruksene. Underveis blir informantene presentert med både egenproduserte illustrasjoner i tillegg til noen skjermbilder fra ekte virtuelle simuleringer for å illustrere ikke bare hvordan det fungerer, men også hvordan det de programmerer påvirker disse simuleringene. Disse er inkludert for å

informere, men også slik at de kan få et klarere bilde av hvordan et system som dette faktisk kan påvirke hvordan opplæring kan utvikles selv av de som ikke har noe programmeringserfaring. Etterhvert som oppgavene blir løst legges stadig flere elementer til i programkoden før de til slutt får en oppgave de skal løse ved å programmere på egenhånd ved å bruke konseptene de har lært. I neste underkapittel vil oppgavesettene presenteres.

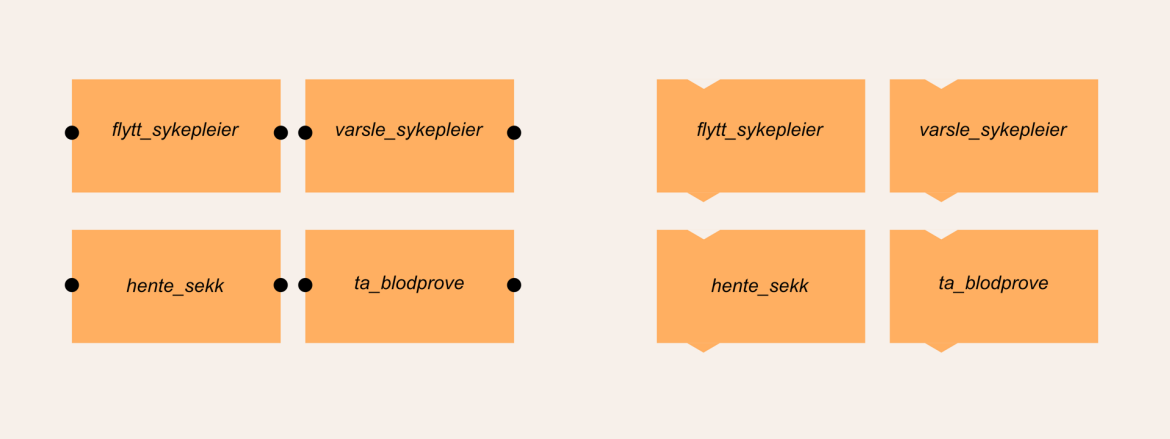
2.4.3 Informasjon og presentasjon i oppgavesettene



Figur 2.5: De forskjellige nodene(v) og blokkene(h) i oppgavesettene

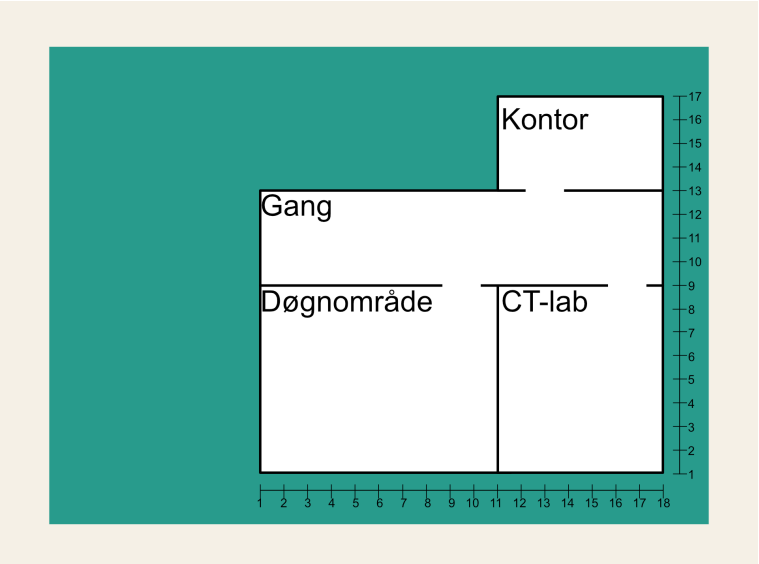
Programmeringsspråkene er fiktive og fokuset ligger heller på forståelse av konsepter og programmering generelt og ikke hvorvidt språket ville kunne vært brukt eller ikke i et ordentlig system. I figur 2.5 finnes en oversikt over de forskjellige blokkene og nodene som tas i bruk i oppgavesettet. På venstre side er nodene, mens på høyresiden ser man blokkene. De oransje nodene/blokkene er der oppgavesettet starter, disse håndterer naturlig språk. Disse inneholder tekst, eksempelvis «ta_blodprove» som gir en indikasjon på hva det tekstlige innholdet i noden/blokken er, i dette tilfellet vil det fullstendige innholdet i denne være teksten «Ta blodprøve av pasient». Språket har også en lyseblå og mørkeblå node/blokk og disse brukes for å gi instrukser til datamaskinen. Den lyseblå kommer i et knippe varianter som håndterer alt fra å flytte objekter som for eksempel personer, notere seg tid eller legge til og fjerne til objekter i et inventory system. Med den mørkeblå noden/blokken håndteres if/else statements. Av de mange konseptene som finnes innenfor programmering ble if/else statements inkludert da dette er et veldig fleksibelt og nyttig verktøy å ha i verktøykassen samtidig som det ikke er så altfor vanskelig å forstå. Hvorvidt det hadde vært fornuftig å introdusere flere programmeringskonsepter eller ikke for å få en helhetlig vurdering av om det faktisk er forståelig for uerfarne programmerere å ta i bruk et system som dette eller ikke kan

diskuteres. For at oppgavesettet ikke skulle ta flere timer å løse ble det derfor begrenset til disse tre forskjellige blokkene og nodene.



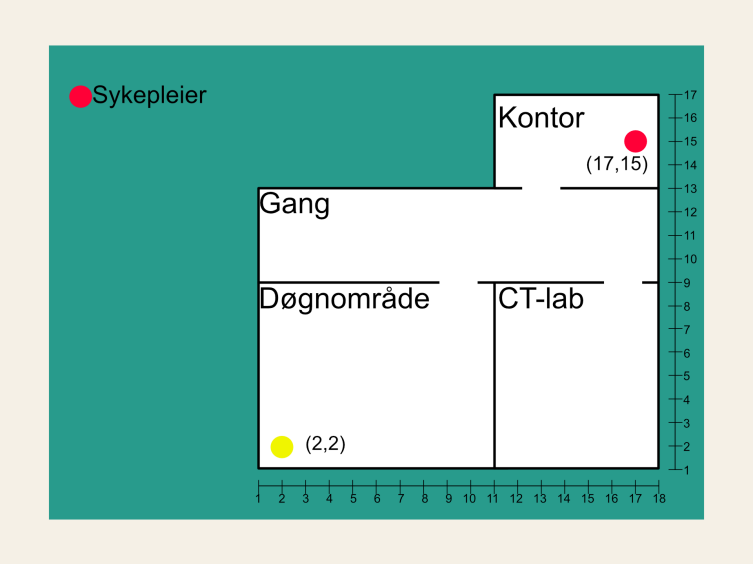
Figur 2.6: Noder og blokker for naturlig språk

Ovenfor finnes fire eksempler på noder og blokker som håndterer naturlig språk. Dette er som nevnt det første som gås gjennom i oppgavesettet, og er de mest grunnleggende og mest brukte elementene gjennom resten av oppgavene. Mye av presentasjonen og informasjonen relatert til disse nodene og blokkene ligger på oppbygning og strukturen av hvordan de settes sammen. Dette er gjort som et forsøk på å gi informantene en bedre forståelse av hva programmering er og hvordan det kan brukes før altfor mange konsepter og elementer blir introdusert slik at denne grunnleggende kunnskapen allerede er på plass når de nye nodene og blokkene blir introdusert. Til denne delen av oppgavesettet følger det kun med en oppgave, alle disse oppgavene vil beskrives i kapittel 2.4.5 og 2.4.6 som tar for seg resultatene.



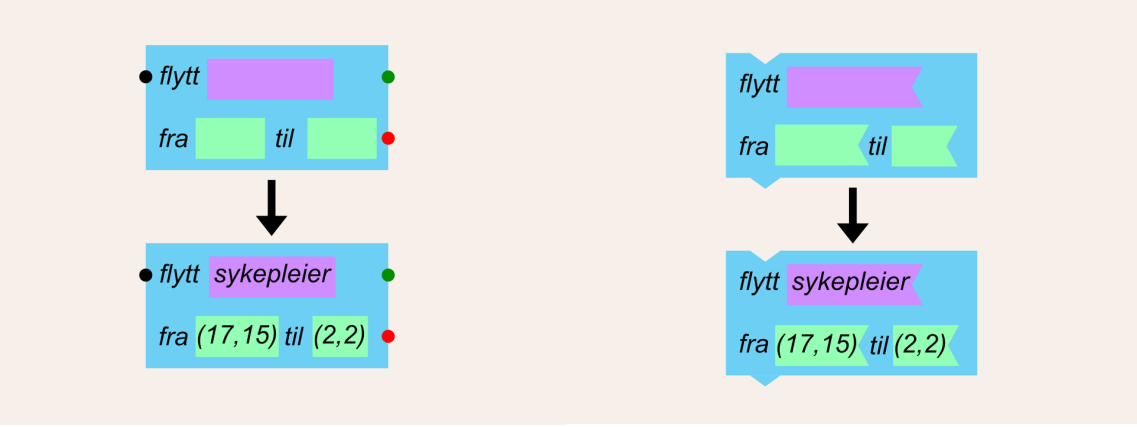
Figur 2.7: Kart laget for oppgavesettet

For å få en introduksjon til hvordan man kan flytte objekter blir informantene presentert med et kart og et koordinatsystem. Her blir det også lagt frem hvordan mennesker og datamaskiner leser informasjon forskjellig, eksempelvis ved å si at mennesker klarer å se hvor for eksempel kontoret er, mens en datamaskin vil trenge koordinater for å skjønne hvor ting er. Videre i oppgavesettet vil det samme kartet brukes, og etterhvert som mer funksjonalitet blir lagt til vil kartet fylles med flere markører og gjenstander. Dette er enkelt og greit for å respektere tiden til informantene ved at de ikke trenger å «lese» et nytt kart hver gang det brukes i eksempler, og at fokuset er rettet på det vi legger til i kartet. Det eneste unntaket fra dette er to av oppgavene hvor de får presentert nye kart og tilhørende oppgaver.



Figur 2.8 Kart med sykepleier for å illustrere hvordan man flytter objekter

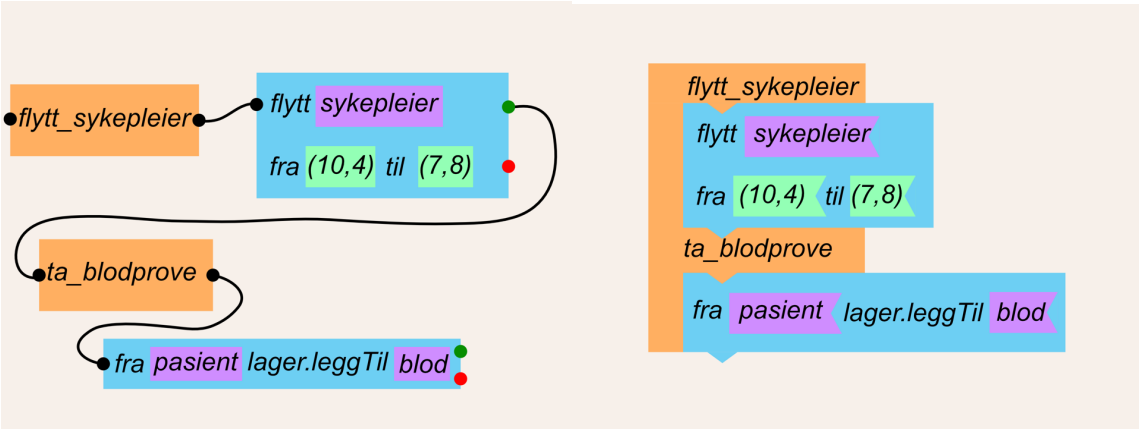
Neste node og blokk blir så introdusert i et eksempel hvor kartet har fått en markør som representerer en sykepleier og markør som representerer lokasjonen sykepleieren skal sendes til i rødt og gult.



Figur 2.9 Node(v) og blokk(h) brukt for å flytte objekter

Disse lyseblå nodene og blokkene er som nevnt brukt for å gi datamaskinen instrukser for hvordan den skal flytte objekter, håndtere tid, legge til objekter i et inventory-system og lignende. De lilla feltene blir beskrevet som en nedtrekks meny som skal inneholde en oversikt over alle objektene tilgjengelig i scenen det jobbes i. I dette tilfellet hvor det eneste objektet i scenen er en sykepleier, så vil dette være det eneste objektet som kan velges. De grønne feltene er tekstfelter som kan fylles ut med koordinater, variabelnavn og lignende. På dette punktet i det nodebaserte oppgavesettet poengteres det også at de forskjellige portene, altså de fargede punktene på nodene ikke skal tenkes på helt enda da de ikke tas i bruk før litt senere. Denne avgjørelsen ble tatt da det ble sett på som nødvendig å nevne slik at de som svarer på oppgavene ikke føler at informasjon blir utelatt eller tar det som en selvfølge at de skal forstå dette på egenhånd. I denne delen av oppgavesettet er det to oppgaver som tar for seg de lyseblå nodene og blokkene, en beskrivelse av alle oppgavene finnes i kapittel 2.4.4.

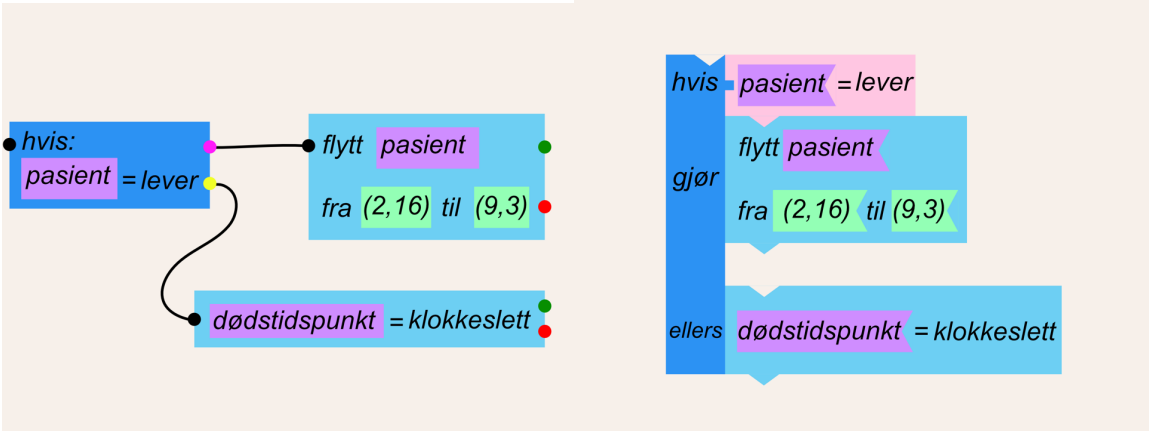
For å få fokuset litt bort fra noder og blokker blir en del av oppgavesettet dedikert til informasjon om hvordan det informantene frem til nå har blitt presentert vil kunne påvirke virtuelle simuleringer. En illustrasjon hentet fra en virtuell simulering med skjermvalg og et relevant miljø vises frem samt en medfølgende forklaring om hvilke elementer på skjermen de forskjellige blokkene representerer. På denne måten kan informantene se resultatet av det de koder, noe som forhåpentligvis fungerer som en motiverende faktor.



Figur 2.10 Illustrasjon for hvordan man kombinerer naturlig språk og kode

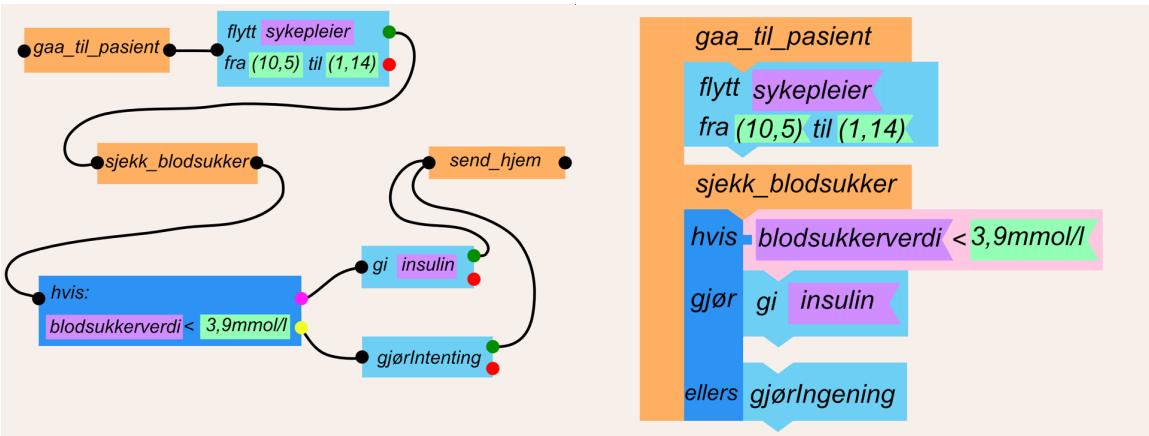
Videre følger det informasjon om hvordan man kobler de oransje og lyseblå nodene og blokkene sammen. Med informasjonen som har blitt presentert frem til nå er det mulig å programmere et enkelt hendelsesforløp. Her begynner også de største forskjellene på de to oppgavesettene å komme frem, og det er herfra og ut de største forskjellene i resultatene vil komme tydelig frem. Nodene kobles naturlig nok sammen med et nettverk av tråder som går

mellom portene på nodene mens blokkene kobles sammen som et puslespill, og selv om de inneholder den samme informasjonen er forskjellene på oppbygningen betydelig nok til at det vil være tydelige forskjeller i forståelse av de to tilnærmingene. Informantene får muligheten til å løse kun et av oppgavesettene, så de vil ikke få muligheten til å sammenligne dem side om side, noe som muligens kan gi mer unike og genuine meninger om hva de liker og ikke liker med hver tilnærming. Oppgavene herfra og ut krever også tekstlige svar, slik at besvarelsene tillater personlige meninger og tolkninger. Ved at besvarelsene er anonyme vil de som gjennomfører oppgavesettene kunne ytre sine meninger uten å være redde for at det setter dem i et dårlig lys.



Figur 2.11 if/else statements med noder(v) og blokker(h)

Siste del av oppgavesettet tar for seg if/else statements. Dette er et konsept som var relativt enkelt å både forstå og forklare ved å bruke gode eksempler samtidig som det er et svært kraftig og nyttig verktøy å ha i et språk som dette. Her er det ikke bare forskjell på selve nodene og blokkene, men også hvordan man bygger videre på disse. Dette er fordi output til betingelsene i den nodebaserte varianten kan sendes videre til enten hver sin node eller samme node, mens man på den blokkbaserte varianten er begrenset av den fastlåste «bygge ovenfra og ned»-strukturen som naturlig nok følger med når man bygger med blokker. Her blir man isteden nødt til å bake ekstra funksjonalitet inn mellom blokkene som allerede er der, noe som potensielt kan føre til uoversiktlig kode. I dette oppgavesettet er det likevel ikke nødvendig å finne noen kreative løsninger i sammenheng med if/else statements da oppgaven som skal løses til slutt kun krever det de får presentert underveis.



Figur 2.12 Alle komponentene satt sammen

Etter å ha satt sammen alt som har blitt lært gjennom oppgavesettet vil koden for eksempel kunne se ut slik som på illustrasjonen ovenfor. Den ekstra oransje noden som sier «send_hjem» er kun inkludert for å vise frem hvordan output fra betingelsene kan gå. Det er omtrent så store kodeeksempler som forventes som besvarelser på den siste oppgaven. Disse to eksemplene illustrerer den samme funksjonaliteten i kode, men her kommer forskjellen i oppbygningen tydeligere frem. Denne siden skal funke litt som inspirasjon før siste oppgave skal besvares. Informantene har også mulighet til å gå tilbake til denne siden mens de løser oppgave 6 skulle de ønske det.

2.4.5 Besvarelser på digital datainnsamling

Den digitale datainnsamlingen varte fra 15.mai til 15.september 2021. Etter å ha samlet inn data fra både aktive sykepleierstudenter og ansatte ved Høgskolen i Østfolds avdeling for helse og velferd har følgende antall besvarelser blitt samlet inn til de to oppgavesettene. Merk at denne tabellen kun inkluderer digitale besvarelser. Resultatet fra den fysiske innsamlingen presenteres i kapittel 2.5.1.

Nodebasert programmering	Blokkbasert programmering
5	4

Tabell 2.1 Antall besvarelser på digital datainnsamling

Den første oppgaven som besvares i begge oppgavesettene er identisk og vil ikke ta for seg forskjellene i programmeringen, men heller fokuserer på hvorvidt folk ser behovet for en slik løsning eller ikke. Besvarelsene virker utelukkende positive til en løsning som gir helsepersonell muligheten til selv å kunne tilpasse opplæringsverktøyene. Dette er en

motiverende faktor som nok en gang poengterer viktigheten av å dedikere mer tid og ressurser på et større prosjekt som tar for seg sluttbrukerverktøy for opplæring i helsesektoren. Systemer som dette er ikke nødvendigvis noe de tenker på selv eller føler de mangler, men alle besvarelsene stiller seg positive til nye digitale løsninger hvor de selv har mer kontroll. Et poeng som også dukket opp i noen av besvarelsene var at det kanskje vil være mer lønnsomt for mindre institusjoner med slike løsninger da de største sykehusene og slikt gjerne allerede har ansatt en større gruppe med IT-personell. Nedenfor finnes noen utdrag fra besvarelsene.

«*Jeg syntes dette gir mening, og er ett veldig bra alternativ for mindre sykehus og andre sektorer innenfor helse og medisin*»

«*Dette gir mening og behovet er der. Enklere digitale løsninger innenfor helsevesenet er nødvendig, da man ofte i dag må kontakte IT-support, noe som krever både tid og penger*»

«Ja dette kan være ressursbesparende, men avhenger av brukergrensesnitt»

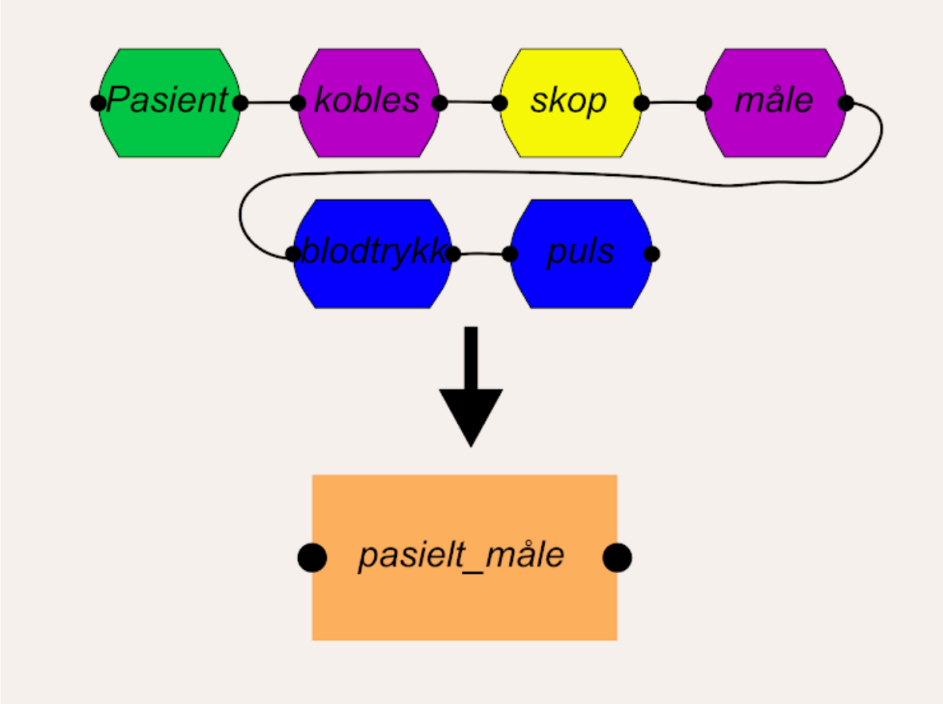
De resterende oppgavene er spesifikke for hvert oppgavesett, og et sammendrag på hver enkel av disse vil finnes nedenfor i delkapittel 2.4.6 og 2.4.7.

2.4.6 Nodebasert oppgavesett

Dette delkapittelet vil kun være et kortfattet sammendrag av oppgavene og de innsamlede dataene tilhørende det digitale oppgavesettet for nodebasert programmering. Diskusjon og refleksjon over resultatene vil finnes i kapittel 4.

Oppgave 1:
Den første oppgaven på begge oppgavesettene er like mye for å illustrere for brukeren hvordan nodene og blokkene er konstruert mer enn å sjekke forståelsen for programmeringskonsepter. Oppgaveteksten finnes nedenfor. I tillegg til denne informasjonen blir også informantene i forkant informert om betydningen av fargekodene på de forskjellige nodene. Dette har likevel ingen betydningen for innholdet i nodene annet enn at det er ment som en pekepinn på hvilken kategori hvert ord hører til.

I denne oppgaven har vi laget den oransje noden du kan se nedenfor som heter "pasient_måle". Innholdet i denne er satt sammen av de seks fargede nøkkelordene du kan se i bildet. Hva tror du "pasient_måle" skriver til brukeren?



Svaret ditt

Figur 2.13 Oppgave 1 i det nodebaserte oppgavesettet

Jevnt over virket forståelsen for hvordan innholdet i blokkene settes å være relativt god. Noe som gikk igjen med enkelte besvarelser var at enkelte nøkkelord ble utelatt. Eksempelvis:

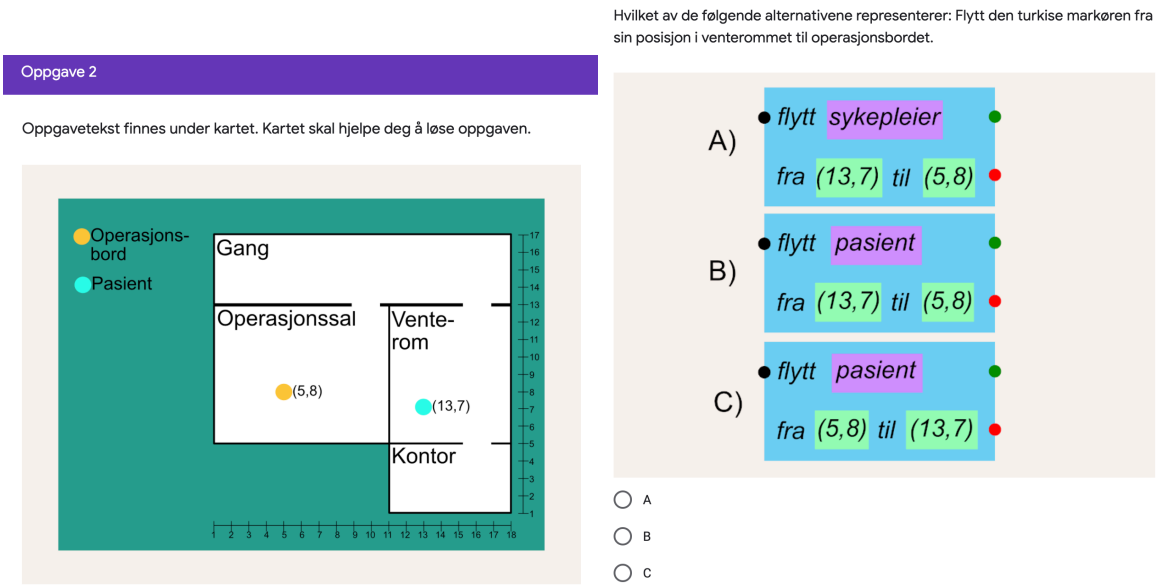
«Mål blodtrykk og puls til pasient.»

Andre besvarelser skjønte hva oppgaven gikk ut på, men svarte ikke nøyaktig på det oppgaven spør om. Besvarelser som dette viser likevel en grad av forståelse for hva innholdet er. Alle nøkkelordene er her med i besvarelsen, men istedenfor å kun skrive setningen denne blokken skriver til brukeren, inneholder heller besvarelsen et sammendrag av hva innholdet er. I retrospekt kunne muligens besvarelser som dette vært unngått dersom oppgaveteksten hadde vært formulert annerledes. Se eksempel:

«Resultatet av målingene som blir gjort. Altså blodtrykk og puls når pasient koblet til skop»

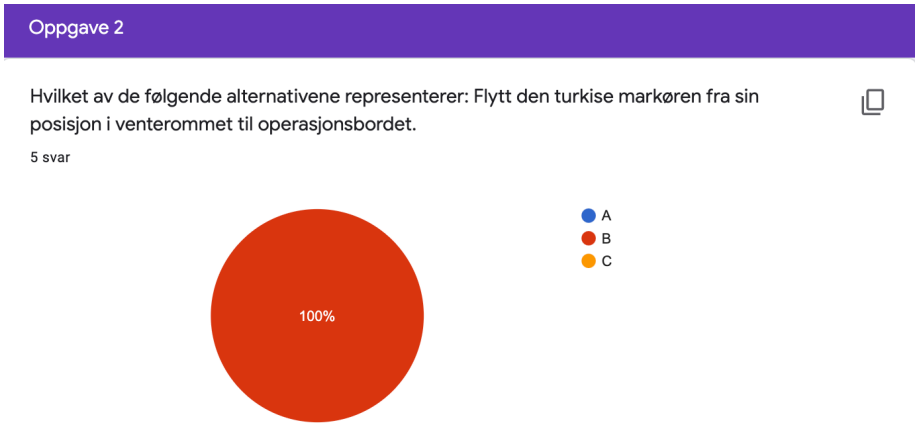
Oppgave 2:

Fra oppgave 1 til oppgave 2 har informantene lært om blant annet hva slags informasjon maskinen trenger for å forstå instruksene den blir gitt. Oppgaven inneholder et kart hvor informanten blir bedt om å velge riktig node som representerer handlingen å flytte pasienten til operasjonsbordet som er representert av den gule markøren.



Figur 2.14 Oppgave 2 i det nodebaserte oppgavesettet

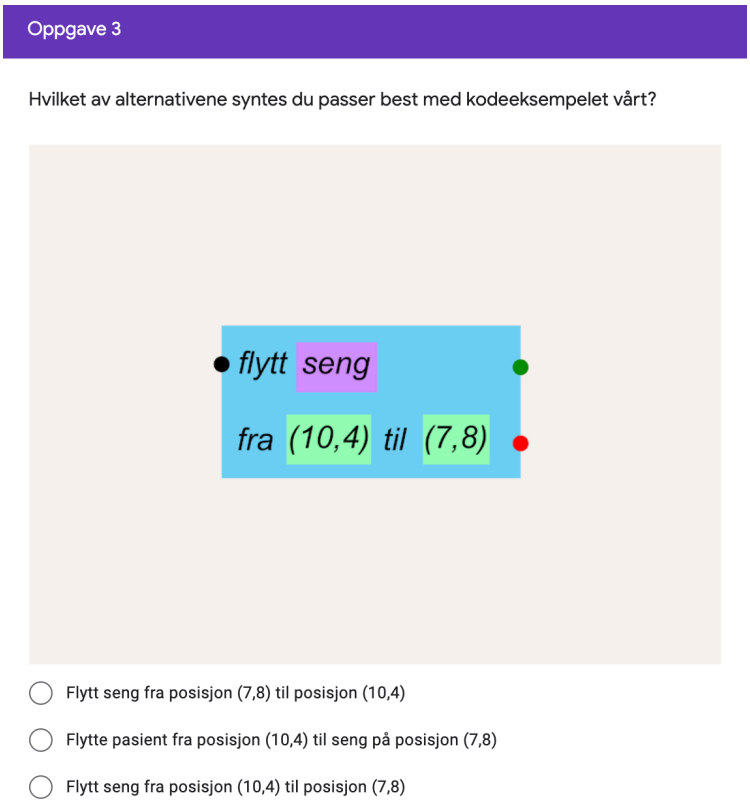
Dette er en flervalgsoppgave, og denne oppgaven er korrekt besvart i alle besvarelsene.



Figur 2.15 Kakediagram over besvarelser på oppgave 2

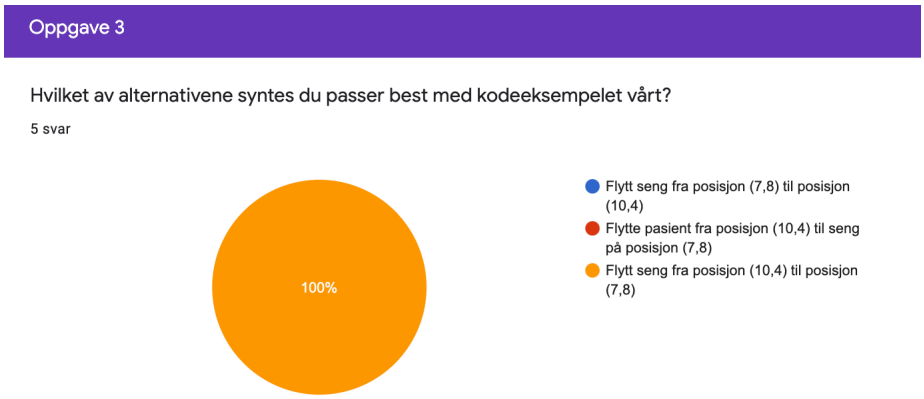
Oppgave 3:

Denne oppgaven er også en flervalgsoppgave, og kommer som en direkte oppfølger til forrige uten noe mer informasjon. I denne oppgaven blir informantene bedt om å velge beskrivelsen som passer best til en node istedenfor å velge hvilken node som passer beskrivelsen. Motsatt av forrige oppgave med andre ord.



Figur 2.16 Oppgave 3 i det nodebaserte oppgavesettet.

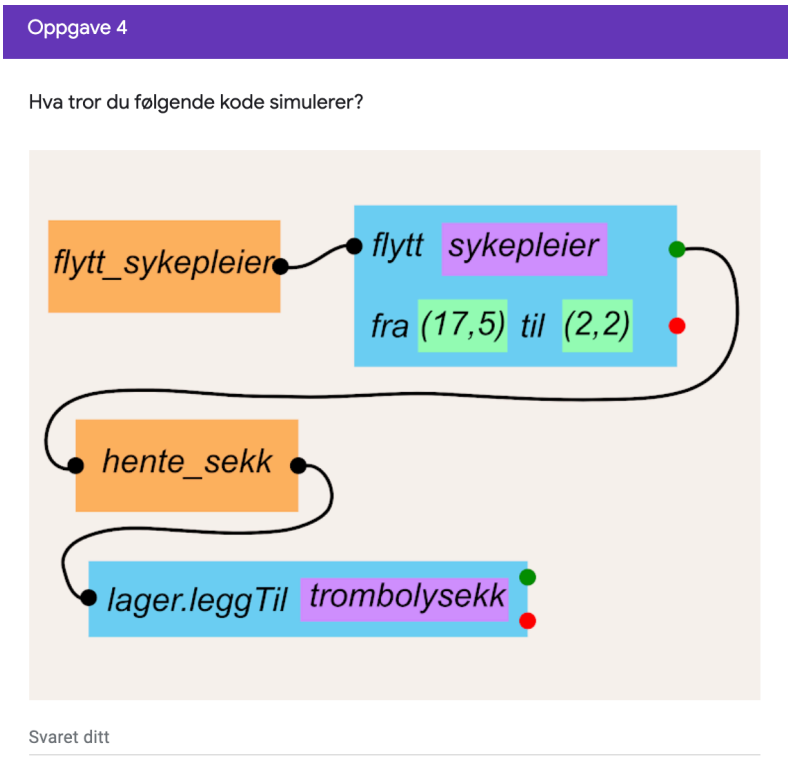
På lik linje med forrige oppgave var også denne oppgaven noe brukerne skjønte, og alle besvarelsene hadde riktig alternativ.



Figur 2.17 Kakediagram over besvarelser på oppgave 3

Oppgave 4:

Nå som informantene har fått info om hvordan man kommuniserer med naturlig språk til brukeren og hvordan man kan kommunisere med maskinen blir de nå bedt om å sette denne informasjonen sammen.



Figur 2.18 Oppgave 4 i det nodebaserte oppgavesettet

Besvarelsene reflekterer hendelsesforløpet som koden beskriver. Likevel er det i likhet med oppgave 1 detaljer som blir utelatt. I tillegg tar noen av besvarelsene seg noen friheter når det kommer til tolkninger som ikke nødvendigvis samsvarer med det koden egentlig gjør. Se eksempel nedenfor.

«Sykepleier skal gå fra eks pasientrom til skyllerom og hente trombolysesekk»

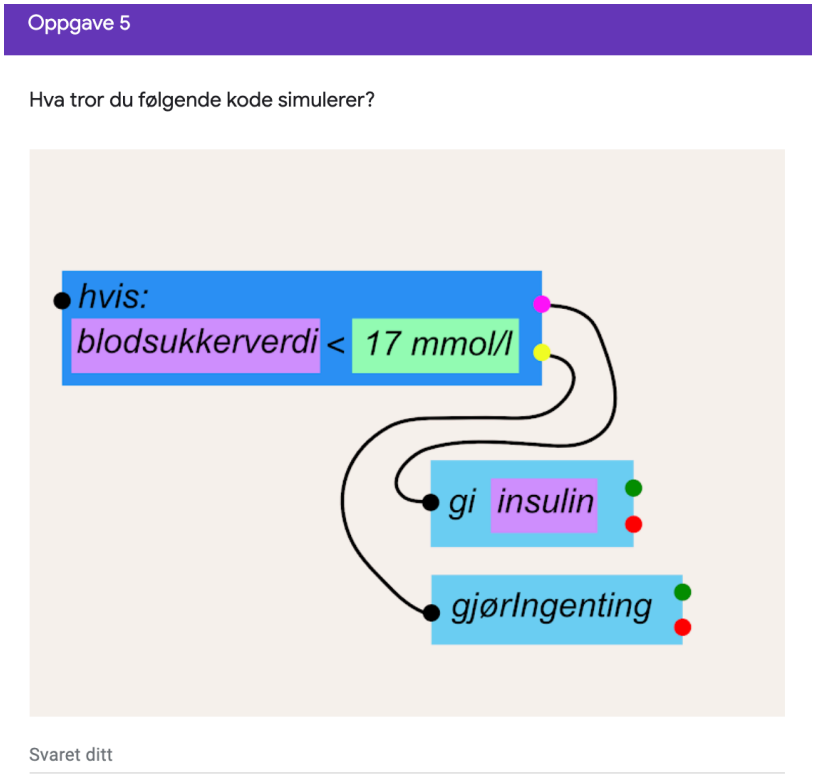
«Sykepleier henter trombolysesekk.»

Det var også besvarelser som beskriver beskjedene til maskinen godt, men det er gjennomgående her at brukerne glemmer de oransje nodene og det totale bilde av både naturlig språk og kommunikasjonen med maskinen. I eksempelet nedenfor er fortsatt lagersystemet misforstått.

«At sykepleiere skal flyttes fra ett punkt til ett annet, hente en sekk på lager og legge til trombolysekk»

Oppgave 5:

Denne oppgaven sjekker om informantene har forstått hvordan if/else statements fungerer.



Figur 2.19 Oppgave 5 i det nodebaserte oppgavesettet

Forståelsen virket jevnt over god i besvarelsene i den forstand at konseptet ble forstått. Det var likevel enkelte som blandet hva som var var if og else, dette til tross for at oppgaven ble laget med en verdi såpass høy som 17 mmol/l som burde si en nåværende eller fremtidig sykepleier om man skal gi insulin eller ikke.

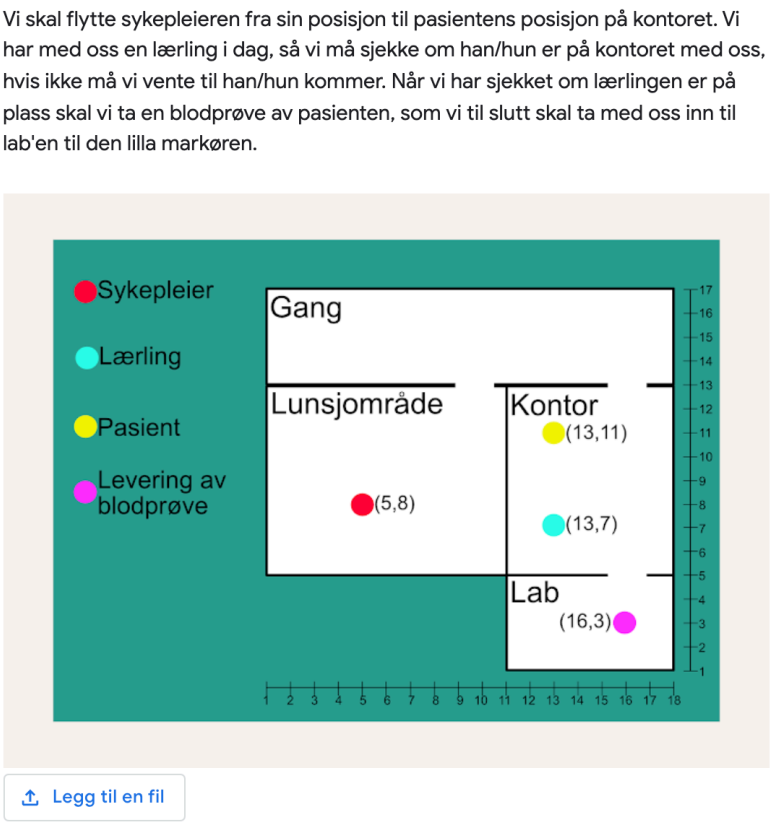
«Hvis BS er over 17mmol/l skal deg gis insulin, hvis under ikke gi insulin»

«Hvis blodsukkerverdien er mindre enn 17mmol/l så gi insulin. Hvis ikke så gjør ingenting.»

Oppgave 6:

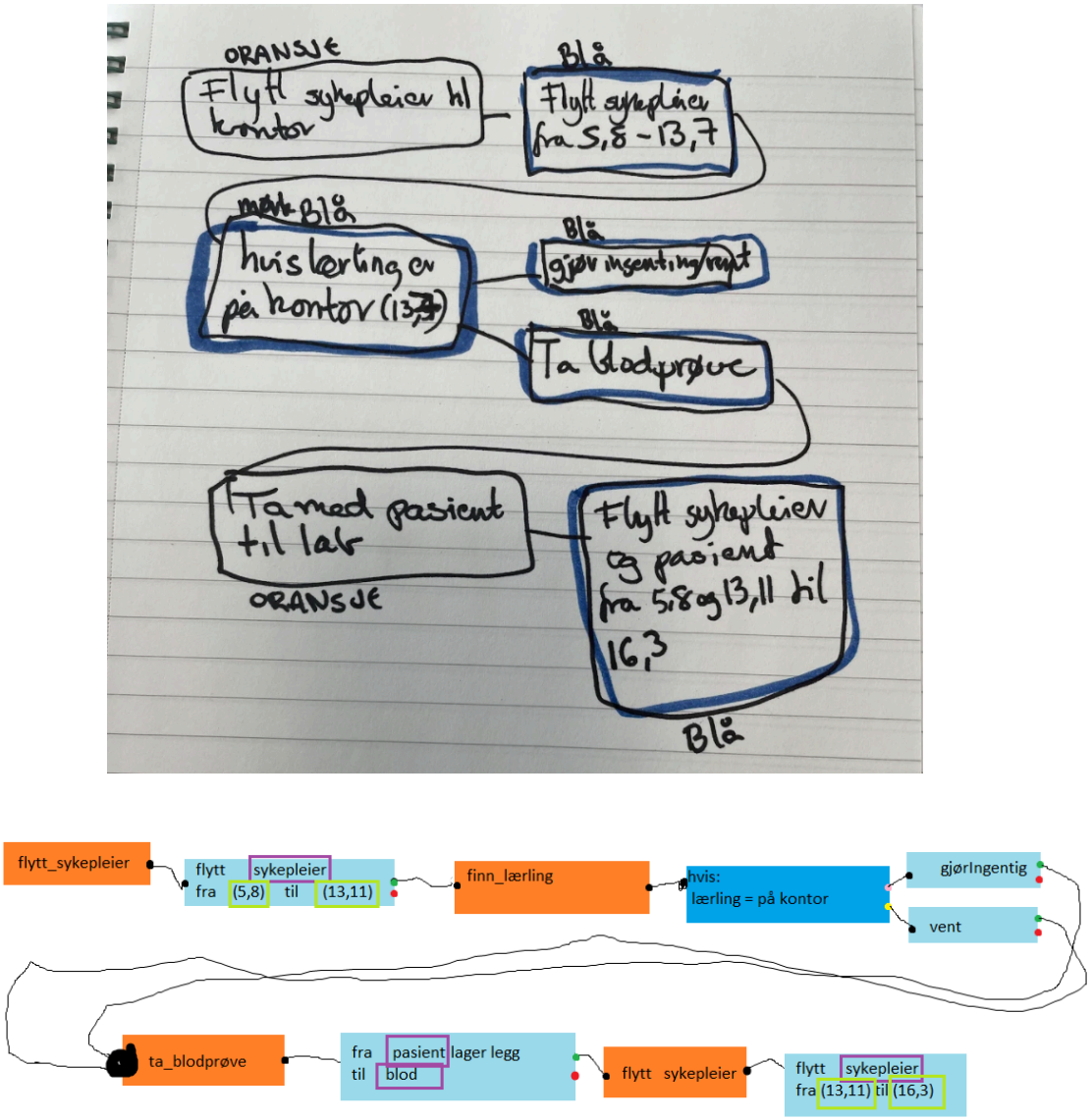
Den siste oppgaven i oppgavesettet er også den mest omfattende. Informantene blir her bedt om å programmere på frihånd med penn og papir, i tegneprogrammer på datamaskinen sin

eller andre metoder de foretrekker. Dette skal til slutt lastes opp. De blir presentert med et scenario de skal gjenskape med de elementene av nodebasert programmering de har lært frem til nå.



Figur 2.20 Oppgave 6 i det nodebaserte oppgavesettet

Besvarelsene til denne oppgaven var som forventet av varierende kvalitet både hva gjelder innsats og forståelse. Noen satte seg ned med penn og papir for å tegne detaljerte noder, mens andre gjorde det enkelt og tegnet i Snapchat på telefonen sin. Dette er selvfølgelig å forvente med tanke på arbeidsmengden de nå har lagt ned i å gjennomføre oppgavesettene, men det gjør det noe utfordrende å sammenligne forståelsen når det er så stor variasjon i kvaliteten. Nedenfor finnes eksempler på to besvarelser.



Figur 2.21 To besvarelser på oppgave 6 fra det nodebaserte oppgavesettet

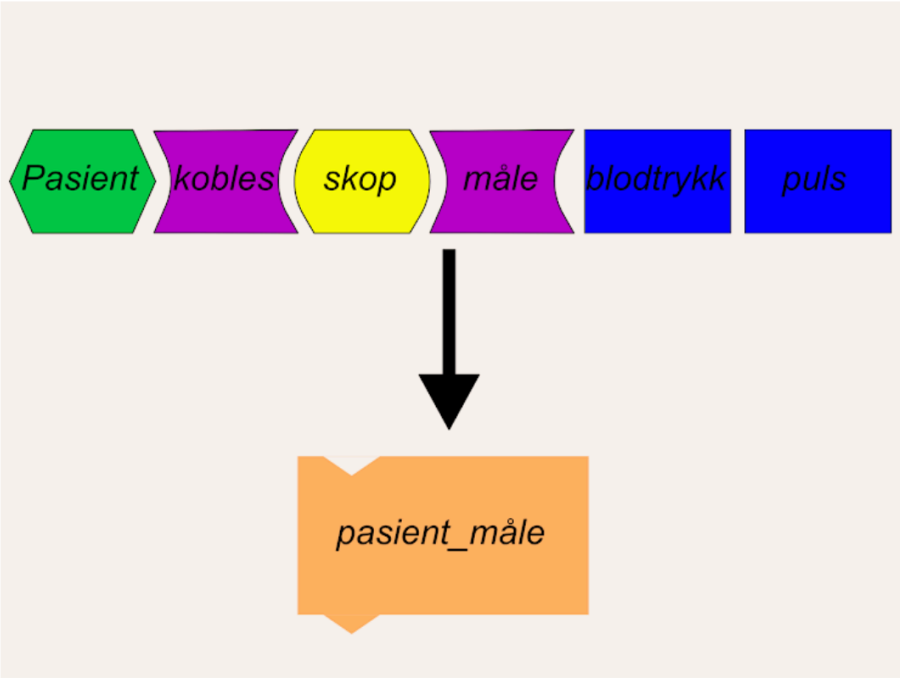
2.4.7 Blokkbasert oppgavesett

Dette delkapittelet vil være likt strukturert som 2.4.6, men tar for seg de blokkbaserte oppgavene. Oppgavesettet har en besvarelse mindre enn hva det nodebaserte har, men resultatene forteller noe om forskjellen i forståelsen likevel. Se kapittel 3 og 4 for mer diskusjon om resultatene. Den samme informasjonen finnes i begge oppgavesettene frem til blokkene og nodene skal kobles sammen. Informantene har derfor relativt likt grunnlag for å forstå hvordan de første oppgavene fungerer, da henholdsvis oppgave 1-3 inneholder samme informasjon. Disse vil derfor ikke bli beskrevet så utfyllende som i kapittel 2.4.6 for å unngå gjentagelse.

Oppgave 1:

På lik linje med det nodebaserte oppgavesettet er den første oppgaven inkludert for å sjekke forståelsen av brukerne i den forstand at den viser hvordan innholdet i blokkene er satt sammen.

I denne oppgaven har vi laget den oransje blokken du kan se nedenfor som heter "pasient_måle". Innholdet i denne er satt sammen av de seks fargede nøkkelordene du kan se i bildet. Hva tror du "pasient_måle" skriver til brukeren? *



Svaret ditt

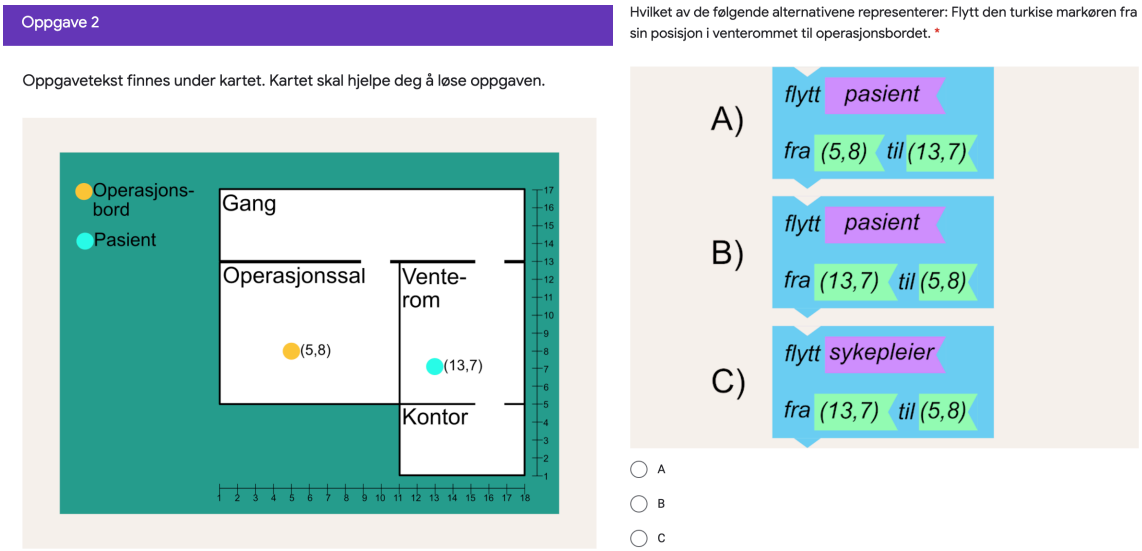
Figur 2.22 Oppgave 1 i det blokkbaserte oppgavesettet.

Nok en gang gjenspeiler besvarelsene en noe unøyaktig oppgavetekst som gir oss svar som er ganske likt utformet hvor informantene, istedenfor å skrive teksten den oransje blokken inneholder, skriver informantene hva de tenker. Det var likevel en besvarelse som traff bra, se eksempler nedenfor.

«jeg tror at den oransje blokken skriver til brukeren at en (helsepersonellet) skal bruke et stetoskop(?) for å måle blodtrykk og puls, evt. bare "Måle blodtrykk og puls"»

«Man skal koble pasienten opp til skopet for å måle blodtrykk og puls.»

Oppgave 2:

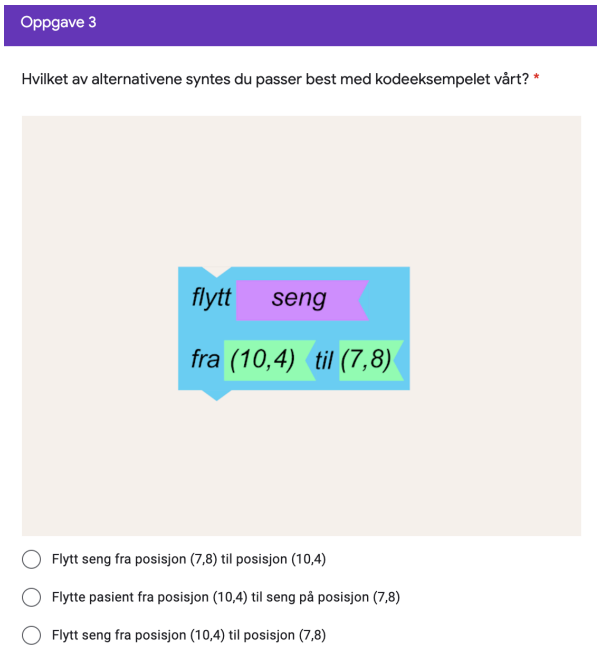


Figur 2.23 Oppgave 2 i det nodebaserte oppgavesettet.

Også her virker det som flervalgsoppgavene har god forståelse. Alle besvarelsene her har valgt riktig alternativ, som er alternativ B.

Oppgave 3:

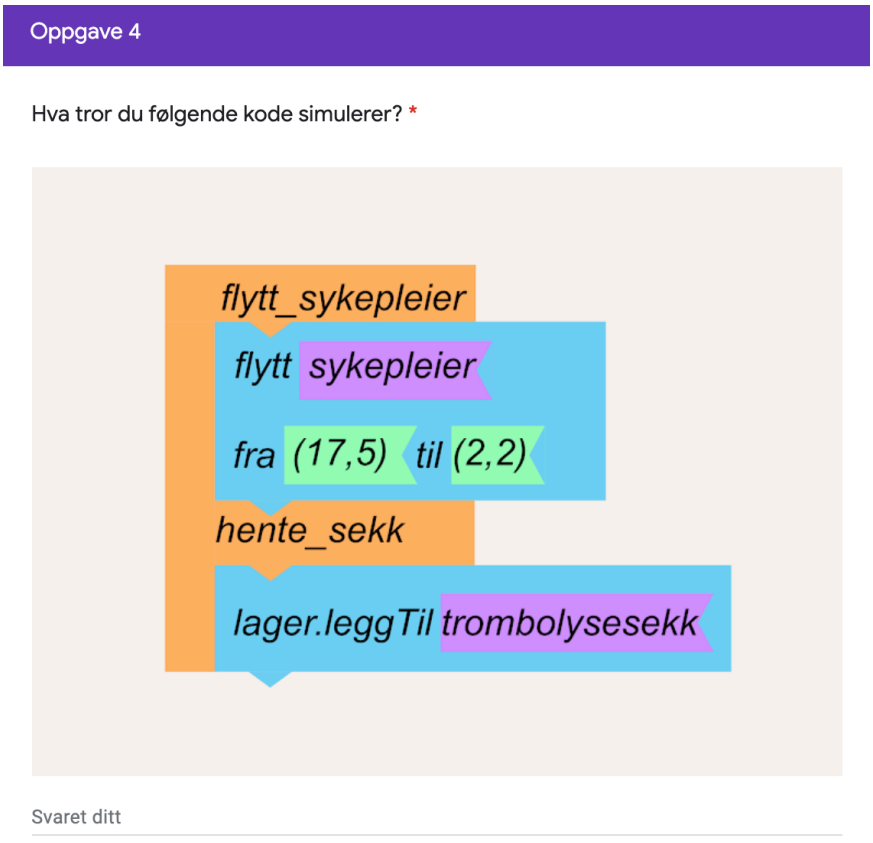
Oppgaven og informasjonen brukerne sitter med her er nok en gang ganske lik de nodebaserte settet, og kan derfor leses mer om i foregående kapittel.



Figur 2.24 Oppgave 3 i det blokkbaserte oppgavesettet

Også her er alle besvarelsene korrekte. Alle har valgt alternativ C, som er riktig svar.

Oppgave 4:
I oppgave 4 kommer forskjellene på de blokkbaserte og nodebaserte oppgavene mer tydelig frem. I denne oppgaven skal de tolke kode som er satt sammen av flere blokker, noe som gjør at måten de leser koden, og deres forståelse av hvordan den henger sammen på i stor grad avgjør hvor nøyaktig de klarer å besvare oppgavene. Oppgavene i begge settene spør fortsatt om det samme om det er samme kode som er representert, men koden må leses på en helt annen måte.



Figur 2.25 Oppgave 4 i det blokkbaserte oppgavesettet

Forståelsen og rekkefølgen koden leses i er som blir sett på når de to tilnærmingene blir sammenlignet i kapittel 3. En av besvarelsene hadde en litt interessant tolkning av koden de ble presentert med. Vedkommende tolket det som at koden var grensesnittet man til slutt satt igjen med, og i sin forklaring av koden refererte vedkommende til blokkene som knapper man kunne trykke på. Flere eksempler ligger nedenfor.

«Hvis man trykker på den øverste oransje blokka vil man flytte objektet "sykepleier" fra posisjon 17,5 til 2,2. Deretter trykker man på den nederste oransje blokka, og tar med seg sekken på veien.»

«Jeg skal flytte sykepleier til lageret (til koordinat 2,2). På lageret skal sykepleier hente trombolysesekk.»

Oppgave 5:
Dette er oppgaven om if/else statements. I motsetning til den nodebaserte tilnærmingen som utgreiner seg til to separate noder, så leses de blokkbaserte if/else statementene ovenfra og ned slik som vi gjør med resten av koden.

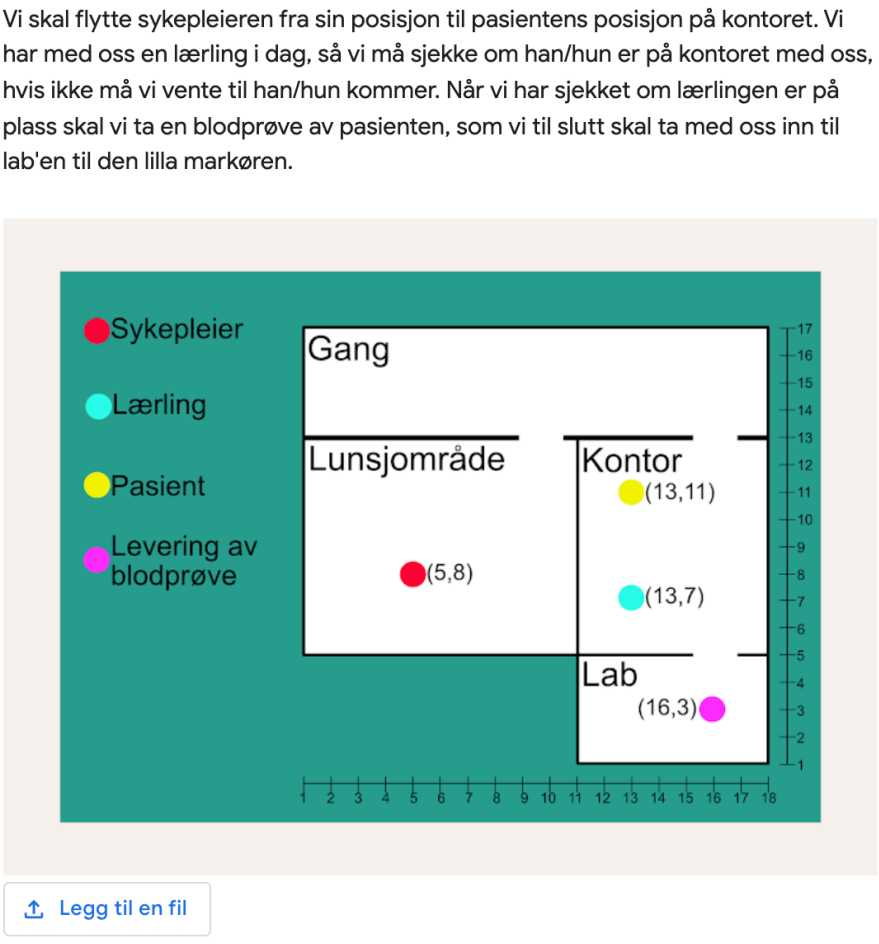
Figur 2.26 Oppgave 5 i det blokkbaserte oppgavesettet

Også her er det noen som blander hva som skjer om utsagnet er sant eller ikke. Alle besvarelsene virker å forstå konseptet med at man har et utsagn med to utfall basert på om utsagnet er sant eller ikke, men utfordringen ligger mer med å forstå om det er «gjør» eller «ellers» som skal velges.

«Hvis blodsukkeret e under 17 mmol/l, gi insulin eller ikke gjør nånn ting»

«Hvis bludsukkerverdien er over 17 mmol/l - enten gi insulin eller ikke gjør noen ting.»

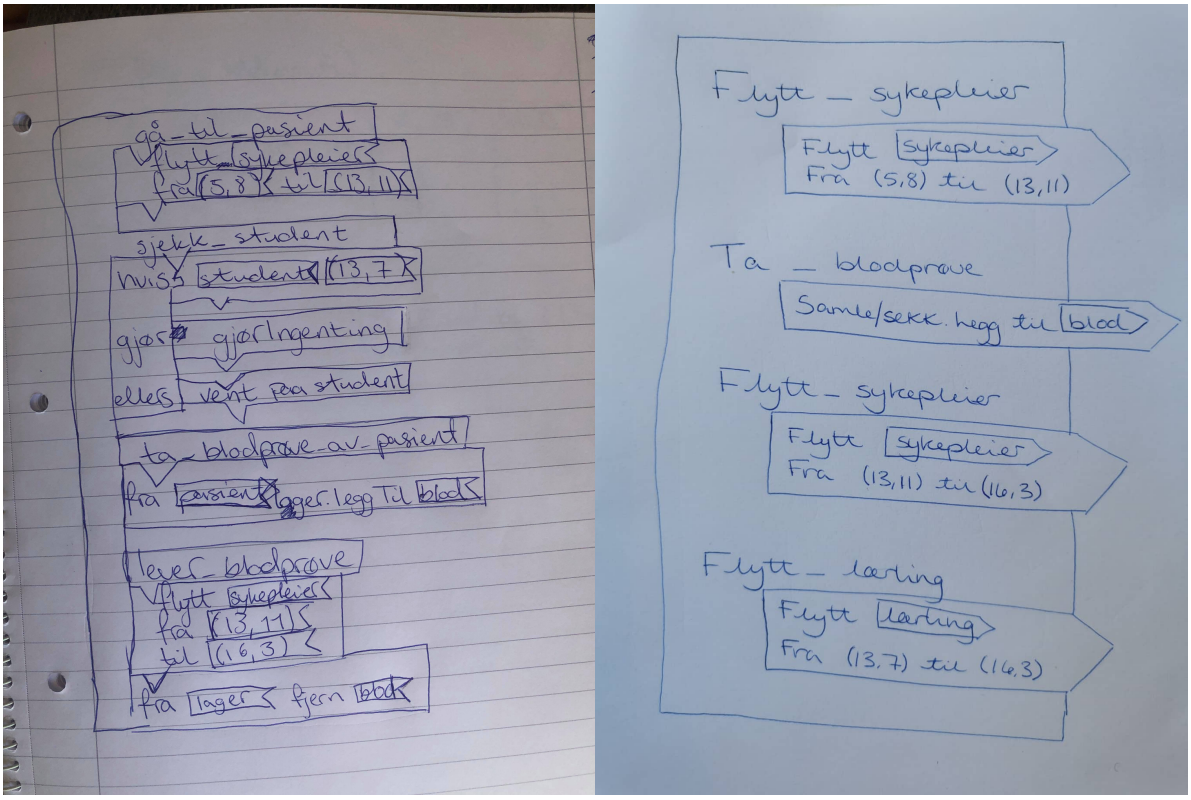
Oppgave 6:



Figur 2.27 Oppgave 6 i det blokkbaserte oppgavesettet. Denne er lik i begge.

Besvarelsene viser jevnt over en grei forståelse, den ene besvarelsen traff veldig godt med løsningsforslaget til oppgavesettet. Denne oppgaven får testet den totale forståelsen av det som har blitt lært frem til nå. I den blokkbaserte varianten er det viktig at man tenker litt på hvordan blokkene er formet i og med at de alle har forskjellige passformer for å passe inn i

hverandre kun i enkelt kombinasjoner. Dette virker å være noe som har blitt litt glemt i besvarelsene, og selv om koden i seg selv kan være ganske riktig blir det likevel feil når man bare har stablet blokker i hverandre uten å ta hensyn til disse reglene.



Figur 2.28 To besvarelser på oppgave 6 fra det blokkbaserte oppgavesettet

2.4.8 Sammendrag av fase 3

Basert på erfaringene fra fase 1 og 2 ble det utviklet to digitale oppgavesett som legger grunnlaget for datainnsamlingen. Disse oppgavesettene inneholder den samme informasjonen og de samme oppgavene, men tar for seg hver sin tilnærming til visuell programmering. I denne fasen ble også første del av datainnsamlingen gjennomført. I tillegg til å gi resultater som kunne hjelpe med å svare på forskningsspørsmålet ga også innsamlingen et innblikk i hvordan den fysiske innsamlingen kunne gjøres. Oppgave 1 i oppgavesettet har for eksempel lite relevans på forståelsen av programmeringen, og blir derfor ikke inkludert i fase 4. Istedenfor å spørre flere informanter om å programmere den siste oppgaven blir denne også endret litt til den fysiske datainnsamlingen. Det blir heller gjort en muntlig gjennomgang hvor informantene blir bedt om å reflektere over hvordan de vil angripe problemet og hvordan de ville løst oppgaven.

2.5 Fase 4 – Testing: Fysisk datainnsamling

Etter hvert som Covid-19 restriksjonene lettet åpnet muligheten seg for å gjennomføre en fysisk datainnsamling. Den siste fasen av prosjektarbeidet tar derfor for seg en datainnsamling i form møter og intervjuer med ansatte fra Høgskolen i Østfolds avdeling for helse og velferd i Fredrikstad. Sammenlignet med fase 3 vil fysiske møter tillate noe veiledning dersom informantene sitte fast eller slite med å forstå noen av konseptene de blir introdusert for. I motsetning til den digitale innsamlingen kunne også gruppen av informanter velges manuelt siden invitasjoner og intervjuer måtte gjennomføres fysisk. Informantene som ble valgt til intervjuene hadde derfor enten tilknytning til simuleringssenteret som administreres av Høgskolen i Østfold eller er faglig ansatte på høgskolens avdeling for helse og velferd. Oppgavesettene var de samme som ble brukt for den digitale innsamlingen.

Målet med denne datainnsamlingen var naturligvis å få samlet inn mer data, da det ikke var tilstrekkelig kvalitativ data samlet inn etter fase 3. I tillegg til å besvare oppgavene, som denne gangen ble gjort muntlig, tilbyr fysiske intervjuer mer innsikt i tankegangen til informantene mens de tolker programmeringen og løser oppgavene. Før oppgavesettene ble presentert ble informantene informert om formålet med prosjektet og hva som var målet med datainnsamlingen. De fikk også muligheten til å stille spørsmål før intervjuene startet. Samtidig var det viktig at de ikke fikk for mye informasjon slik at datainnsamlingen kunne sammenlignes med det som har blitt samlet inn fra de digitale oppgavesettene. Identiteten til informantene er anonymisert i tråd med avtale med NSD. Informantene fikk og signerte en samtykkeerklæring slik at de hadde mulighet til å trekke tilbake samtykket sitt.

2.5.1 Gjennomføring av intervjuer

Disse besvarelse kommer i tillegg til antallet besvarelser som finnes i de to foregående underkapitlene. Den fysiske innsamlingen av data endte til slutt med fem besvarelser, hvorav alle hadde tilknytning til Høgskolen i Østfold sin avdeling for helse og velferd. Tre av informantene jobber i direkte tilknytning til simuleringssenteret, de to siste var faglig ansatte. Den fysiske datainnsamlingen ble gjennomført i uke 43 og 44 i 2021.

Nodebasert programmering	Blokkbasert programmering
2	3

Tabell 2.2 Antall besvarelser på den fysiske datainnsamlingen

Det ble gjort lydopptak av fire av de fem intervjuene, og alle intervjuene ble gjort individuelt. Det siste intervjuet ble gjennomført på Sykehuset Østfold Kalnes i lunsjområdet til de ansatte, og det ble ikke tatt lydopptak da det satt sykepleiere og leger rundt i nærheten som diskuterte sensitiv informasjon. Det ble derfor ført en skriftlig protokoll av intervjuet. Av de fem besvarelsene var det tre som på forhånd visste hva formålet med prosjektet var, mens de to siste gikk inn til intervjuet i blinde. Resultatene vil bli presentert i samme format som i kapittel 2.4. Det vil ikke bli inkludert skjermbilder her da disse allerede finnes i de foregående kapitlene. Oppgave 1 ble hoppet over da denne viste seg å ha liten hensikt etter den digitale innsamlingen.

Oppgave 2 og 3:
Oppgave 2 og 3 er begge flervalgsoppgaver og jevnt over virket dette å være godt forstått, hvor alle endte med rett svar etter litt betenkningstid. Tatt i betraktning at både nodene og blokkene her i utgangspunktet er strukturert helt likt er dette å forvente. Det er mulig at enkelte kunne ha hatt problemer med å besvare denne oppgaven dersom de ikke hadde hatt muligheten til å spørre om hjelp underveis. Med unntak av en besvarelse var alle informantene raske med å finne frem til riktig alternativ. Det var her en mistolkning av hvor pasienten faktisk befant seg, da informanten foreslo at «pasienten er på operasjonsbordet», noe som ikke stemte. Noen sitater fra oppgave 2 finnes nedenfor.

Informant: *«Først vil jeg finne pasient eller sykepleier. Her er det pasient som skal flyttes?*

Meg: *«Her er det den turkise som skal flyttes, så det er pasient.»*

Informant: *«Også vil jeg finne den gule som er operasjonsbordet. Pasienten er på operasjonsbordet, er det ikke sånn da?*

Meg: *«Pasienten er på 13,7 og skal til 5,8».*

Informant: *«Da vil jeg si: Flytt pasient fra 13,7 til 5,8. Så det er nummer 2(alternativ b).»*

Da informantene ble bedt om å besvare oppgave 3 hvor de fikk presentert kode og skulle velge et passende alternativ var det mer utfordrende. Alle endte til slutt med riktig svar, men de trengte mer betenkningstid og spekulasjon for å komme frem til riktig alternativ. Riktig svar på denne oppgaven er alternativ C, men det var likevel to informanter som var raske med å velge alternativ B. Ved å sammenligne dialogen fra oppgave 3 med dialogen fra en av besvarelsene i oppgave 2 kommer det tydelig frem at informantene syntes det er mer utfordrende å forklare hva kode gjør når de ikke har noe visuelt å støtte seg på. Så fort konteksten blir fjernet og de ikke har en visuell representasjon å sammenligne tankene sine med blir de fort usikre. I retrospekt kunne det vært interessant å presentere oppgaven til noen uten alternativene for å se om informantene hadde klart å tenke seg frem til riktig svar.

Informant: *Starter med 10 sekunder stillhet for å tenke. «Så jeg skal flytte sengen? Da er det jo alternativ nummer 2(B). Flytt pasient fra posisjon 10,4 til seng på posisjon... Men det er jo.. Eller vent..»*

Meg: *«Hvordan er det du tenker da?»*

Informant: *«Nei, nei, nå er jeg jo litt sånn her.. Jeg vil jo.. Vi skal jo flytte pasienten... Flytte.. Flytte seng det går jo ikke for der er det jo ingenting. Der er det ingen kode. Men vi skal jo fra 10,4 til 7,8. Da må det jo være: Flytt seng fra posisjon 10,4 og til seng på posisjon 7,8? Men det kan likevel være at.. Den der var litt mer vanskelig. Det her er jo for å sjekke om det er enkelt å bruke, da må jeg tenke da.»*

Meg: *«Hvis du starter på toppen og jobber deg nedover da?»*

Informant: *«Okay. Jeg skal flytte på sengen. Fra 10,4 til 7,8. Da må det jo være alternativ C?».*

Oppgave 4

I oppgave 4 blir informantene bedt om å beskrive med egne ord hva de tror koden de får presentert simulerer. Oppgaven inneholder både de oransje nodene/blokkene og de lyseblå, de skal altså kombinere naturlig språk med informasjon til maskinen. Skjermbilder av hele oppgaven finnes i kapittel 2.4.6 og 2.4.7, dette gjelder også resten av oppgavene. Her kommer de første ordentlige utfordringene frem, som for øvrig gjelder besvarelser på begge

oppgavesettene. Denne utfordringen gjelder hovedsakelig det å skille naturlig språk og språket til datamaskinen fra hverandre. Spesielt i de blokkbaserte besvarelsene er det også noe utfordrende for informantene å skjønne rekkefølgen man leser koden i. Det kom også frem at en av informantene frem til nå hadde misforstått hvordan koden de hadde lært påvirket simuleringene, noe som kommer frem i en av de blokkbaserte besvarelsene. Nedenfor finnes det en besvarelse fra hvert av oppgavesettene.

Nodebasert:
Informant: «*Det er jo at sykepleier skal gå fra et rom til et annet for eksempel å hente en sekk.»*

Meg: Prøver å fiske etter mer *«Hva er det brukerne ser når vi kjører denne koden? Hva er flyten i hele koden hvis du ser fra node til node.»*

Informant: *«Flytt sykepleier til et eller annet rom og hent sekk.»*

Blokkbasert:
Informant: *«Jeg trodde fra starten at de oransje betydde at vi snakket, ikke at vi trykket på en knapp. At den var mer språklig.»*

Meg: *«Det er typisk tekst i menyvalg eller knapper disse representerer.»*

Informant: *«Jeg trodde at det var snakk om språk, men da fikk jeg klarert det. Hva tror du følgende kode simulerer? Hmm.. Skal vi se.. Vi må jo flytte sykepleier. Er det sånn at jeg skal si noe om oransje eller blå?»*

Meg: *«Ta hele koden som du ser her, så både de oransje og de blå. Hvordan tror du det henger sammen? Hvis du hadde kjørt hele koden her, hva tror du hadde vært resultatet av det?»*

Informant: *«Det er jo helt enkelt, at hun skal hente trombolysesekken. Fra 17,5 til 2,2. Sykepleier skal jo hente trombolysesekken, det er jo egentlig det hun skal. Det er det som er hovedsaken, men hvordan vi kommer dit, det er en annen sak. Dette var ikke så enkelt syntes jeg. Jeg må jo flytte... Om jeg skal først.. Jeg må jo flytte sykepleier sånn at jeg kan hente*

sekken, eller trombolysesekken. Og da må hun jo gå fra et visst sted, om det er fra 17,5 det vet jeg ikke. I utgangspunktet så vet jeg ikke hva 17,5 eller 2,2 er for noe sånn som jeg ser det nå, men jeg skjønner at hun skal hente trombolysesekken.»

I begge besvarelsene som er lagt ved her så blir naturlig språk ignorert. I den blokkbaserte besvarelsen spør til og med informanten om vedkommende også skal inkludere den oransje blokken, for så å la være å inkludere den likevel. Igjen gjenspeiler besvarelsen at informantene sliter med se sammenhengen mellom koden og simuleringen når de ikke har noe visuelt å støtte seg på. Det er uklart for informantene hva koordinatene betyr når man ikke har noen knagger å henge disse tallene på.

Oppgave 5:
Denne oppgaven tar for seg if/else statements. På samme måte som i de digitale besvarelsene var dette et konsept det virket som de forstod. Selv om de forstod konseptet med if/else statements var det noen som tolket det litt feil i den forstand at de la til funksjonalitet som ikke fantes i koden. En informant virket også å misforstå *større enn* eller *mindre enn* og sa heller at dersom verdien var noe annet enn det som ble sjekket som så skjedde *else.* Se eksempel nedenfor:

«Den simulerer en blodsukkermåling, at en sykepleier skal måte blodsukker. Og hvis verdien er 17mmol så skal man gi insulin, hvis den ikke er det, da skal man gjøre ingenting.»

Oppgave 6
I den digitale innsamlingen ble informantene bedt om å tegne for så å laste opp sine besvarelser på den siste oppgaven slik at de kunne kode på frihånd. Da det allerede var samlet inn flere slike besvarelser ble det sett som mer hensiktsmessig å heller føre en diskusjon med informantene hvor de fortalte hvordan de ville angrepet problemet og tenkt seg frem til en løsning. Oppgaveteksten prøver som tidligere nevnt å hinte litt til hvilke blokker som skal brukes hvor, men det er overaskende mange som velger å løse utfordringene på alternative måter. Mer analyse og tolkning av resultatene vil diskuteres i kapittel 3. Nedenfor ligger noen utdrag fra besvarelsene til begge oppgavesettene.

Nodebasert(vedkommende hadde i forkant besvart det digitale oppgavesettet):
«Jeg tenker at den første er kanskje den oransje da er på en måte oppgaven at man skal flytte sykepleier fra et sted til et annet. Bak dette så ligger det en kode, og da er det lilla for sykepleier som skal fra si kontor da til lunsjområdet. Også sjekke om lærling er på plass, da kommer den koden der igjen, så da gjør man det, hvis ikke så venter vi, da gjør vi ingenting på en måte. Jeg tror jeg skjønte den. Eneste jeg kanskje var litt usikker på her var hvor mange aktiviteter eller sånne oransje noder jeg skulle putte innimellom forløpet. Jeg delte det opp etter oppgaver, så jeg tenkte at det første med å flytte sykepleier er liksom en oppgave. Så var jeg litt usikker på om det neste var en oppgave, men jeg tenkte sånn at man skal sjekke om pasienten er der eller ikke, og det er jo en ny oppgave. Det er også «ta blodprøve av pasienten», også genererte jeg dette videre.»

Blokkbasert:
Vedkommende som ga den følgende besvarelsen valgte å se på kartet før han/hun jobbet seg gjennom scenarioet. Dette hadde ikke stort med koden å gjøre, men var en interessant observasjon med tanke på måten de jobber på.
«Da ville jeg tatt sykepleier fra koordinat 5,8 til koordinat 13... til koordinat 13,7 og 13,11 da? Nei, jeg vil flytte den til 13,11. Og da er jeg sammen med lærlingen, eller om vi da skal flytte oss fra 13,11 til 13,7 sånn at både sykepleier, pasient og lærling er sammen. Også flytter vi oss derfra inn til 16,3, også skulle vi tilbake igjen. Da er vi jo i samme rom, men jeg forstår ikke helt... for meg nå så blir dette bare en passering, man går fra et sted til et annet sted, og der ser jeg at det står noen. Men hvordan samhandlingen foregår, hvordan man kan sikre at sykepleieren tar med seg lærlingen og hvordan de tar blodprøven, det har jeg jo ikke skrevet noe om. Sånn som jeg tolker det også så burde vi jo hatt kode som sier at sykepleieren skal innom gangen, også fra gangen og inn på kontoret.

Om jeg tenker tilbake slik som vi har jobbet frem til nå så er det jo flere blokker her nå da. Vi begynner med at vi skal flytte sykepleier, så vi flytter sykepleier som er på lunsjområdet...og da må jeg flytte opp.. ja...For det jeg tenker det er jo at man skal gjøre det enkelt, for det kan ikke være mye tekst for da mister man oss. Jeg tenker at vi starter på første punktet, altså flytte sykepleier fra sin posisjon som er på lunsjområdet nå til pasientens posisjon på kontoret, så fra 5,8 til 13,11. Vi må sjekke om lærlingen er tilstede på kontoret, hvis ikke må vi vente til han/hun kommer. Men jeg har jo flyttet sykepleieren dit til kontoret, så den ser jo at lærlingen er der.» Jeg poengterer at datamaskinen ikke nødvendigvis ser det. «Men da må

det være den der hvis/eller? Hvis eller lærlingen er der, så flytter jeg dit. Vi starter på nytt.. Jeg flytter sykepleier til kontoret, og da ser jeg jo at lærlingen er der, så da må jeg jo kunne gå videre? Så tar jeg jo en blodprøve av pasienten, eller er jeg helt ute å kjører? Jeg tror jeg gjør det litt vanskelig? Jeg må jo ta med meg pasienten til lab'en? Eller må jeg det? Jeg skal jo levere blodprøve der, og den skal leveres fra 13,11 til 16,3. Jeg har nok glemt en del punkter.»

Så fort alt av støtte blir fjernet og de må tolke problemene på egenhånd virker informantene fort overveldet og noe usikre på rekkefølgen de skal gjøre ting i. I eksemplene ovenfor ser vi også at den nodebaserte besvarelsen velger å inkludere blokkene i forklaringen sin i stedet for å liste opp oppgavene som skal gjøres slik som vi kan se i den blokkbaserte besvarelsen.

2.5.2 Sammendrag av fase 4

I siste fase av prosjektarbeidet ble den fysiske datainnsamlingen gjennomført. Datainnsamlingen ble gjennomført i form av intervjuer av ansatte ved Høgskolen i Østfolds avdeling for helse og velferd i Fredrikstad. Det ble totalt samlet inn data fra 5 informanter, hvor den største andelen har tilknytning til simuleringssenteret eid av høgskolen. I intervjuene ble de samme oppgavesettene som i den digitale datainnsamlingen i fase 3 brukt, men det ble gjort noen justeringer på hvordan oppgave 1 og 6 ble håndtert. Det står mer om disse endringene i kapittel 2.4.8. Istedenfor å samle inn riktig eller feil svar lå fokuset i intervjuene mer på hvordan informantene tenkte når de skulle løse oppgavene. Dette ga flere interessante funn, blant annet at naturlig språk ofte blir glemt og hvordan de trekker sammenligninger mellom de fysiske simuleringene de er kjent med og flyten i nodebasert programmering. Disse funnene og fler tas med inn i kapittel 3, hvor resultatene presenteres og forskningsspørsmålet blir besvart.

3 Resultater og besvarelse på forskningsspørsmålet

Etter gjennomføring av de to delene av datainnsamlingen har det blitt samlet tilstrekkelig data til å kunne besvare forskningsspørsmålet. Før datainnsamlingen startet var det ikke gitt at en av tilnærmingene skulle vise seg å komme bedre ut enn den andre. Likevel var det for øvrig en dominant andel av de verktøyene som brukes i dag som benytter seg av blokkbasert programmering. De nodebaserte tilnærmingene virker derimot å være mer dominante i mer

komplekse domener slik som spillutvikling hvor hele kodebasen er bygget opp i et visuelt grensesnitt slik som med Unreal Engine sitt Blueprint Scripting system eller Bolt i Unity. Det kan leses mer om disse verktøyene i kapittel 2.2.

Da datainnsamlingen startet var det en noe skjev fordeling i antall besvarelser mellom de to oppgavesettene, med fordel til den blokkbaserte tilnærmingen. Gjennom startfasen av den digitale innsamlingen var inntrykket at blokkbasert programmering var det beste alternativet grunnet et lavt antall besvarelser. Etterhvert som flere av resultatene kom inn ble denne indikasjonen endret, spesielt på den siste oppgaven hvor de skulle kode på frihånd, da denne oppgaven viste at de nodebaserte besvarelsene ofte virket å bedre gjenspeile de konseptene de hadde lært og hvordan de ble brukt på en fornuftig måte. Noe som gikk igjen i flere av besvarelsene, både fra den digitale og den fysiske datainnsamlingen var at informantene glemte å inkludere naturlig språk, enten delvis eller fullstendig. I de fysiske intervjuene var det mulig å prøve å hinte til at det var noe de hadde glemt ettersom det manglet noe informasjon. I flere av disse tilfellene ble dessverre flere mer fokusert på om de hadde glemt noe i selve koden og logikken sin istedenfor at de faktisk hadde glemt et viktig komponent slik som naturlig språk. Denne mangelen viste seg likevel å være mindre på de nodebaserte besvarelsene. I et av de fysiske intervjuene som ble gjennomført for det nodebaserte oppgavesettet prøvde til og med informanten å komme med forslag til hvordan eksemplene i oppgavesettene kunne forbedres ved å legge inn spørsmål til brukerne med de oransje nodene. Dette var en som ikke hadde erfaring med programmering fra tidligere, noe som viser en god forståelse med tanke på at dette også var første gang vedkommende så grensesnittet.

Flere av informantene til den fysiske innsamlingen har på en eller annen måte erfaring eller tilknytning til systemene høgskolen bruker. I undervisningen bruker de digitale simuleringer, men de har også fysiske rom utstyrt som forskjellige behandlingsrom og lignende man kan finne på sykehus. Disse rommene har speil instruktørene står bak hvor de observerer studentene og kan gi de input over et kommunikasjonssystem. I disse fysiske simuleringene har studentene et hendelsesforløp de skal gjennom, og i et av intervjuene på det nodebaserte oppgavesettet ble det trukket sammenligninger mellom programmeringen og disse fysiske simuleringene. Det ble forklart av informanten som at handlingene og måten de ga instrukser til studentene var ganske likt, og at flyten i koden med at man går fra node til node var litt som en sammenhengende tråd med instrukser vedkommende kunne kjenne seg igjen i. En annen sammenligning som ble nevnt i sammenheng med det nodebaserte oppgavesettet var at

det var rimelig enkelt å følge flyten i koden da det lignet noe på flytskjemaer. Hvorvidt dette er noe som brukes daglig av helsepersonell vil trolig variere avhengig av stilling og hvilken institusjon man jobber på, men det er ikke usannsynlig at man i det minste har sett et flytskjema. Det ble ikke trukket noen slike sammenligninger til tidligere erfaringer i intervjuene eller de digitale besvarelsene i sammenheng med det blokkbaserte oppgavesettet. Det er mulig denne sammenligningen er noe unik for helsepersonell, noe som er greit med tanke på omfanget av dette prosjektet, men denne detaljen vil muligens ekskludere enkelte andre målgrupper.

Basert på resultatene og sammenligningene av disse resultatene satt opp mot hverandre er dette trolig nok data til å kunne besvare forskningsspørsmålet i prosjektet. Denne dataen tilsier at den formen for visuell programmering som på best mulig måte balanserer kompleksitet og brukervennlighet for instruktører som har i oppgave å lage scenariobaserte virtuelle simuleringer som brukes til opplæring av helsepersonell er **nodebasert programmering.**

Som nevnt i kapittel 2.1.3 så foreslår Creswell og Creswell at funnene i kvalitativ forskning presenteres i tre kategorier: forventede funn, overraskende funn og uvanlige/konseptuelle funn. Disse tre kategoriene presenteres nedenfor.

Forventede funn: Det kom ikke som noen overraskelse at målgruppen til dette prosjektet hadde utfordringer med å forstå programmering. Utfordringene knyttet seg ikke nødvendigvis til hensikten og konseptene i programmering, men at de håndterer syntaksen som mer fleksibel enn den egentlig er. Dette skyldes hovedsakelig som de sier at det er såpass ulikt noe de har jobbet med før. Av de 14 besvarelsene jeg fikk totalt var det ingen som hadde noe erfaring med moderne programmering, det nærmeste var noen som hadde skrevet kode i DOS for mange år siden, noe som tilsier at majoriteten i målgruppen aldri har skrevet programkode.

Overaskende funn: Måten informantene angrep problemene på og valgte å løse oppgavene var noe interessant og overaskende. Istedenfor å se på hvilke aktiviteter som skal gjennomføres og hvilke komponenter som trengs for å gjennomføre disse aktivitetene så enkelte på hele problemet og prøvde å løse flere eller alle delene samtidig. Dette førte som nevnt tidligere til at valgene av blokkene og nodene ikke nødvendigvis utnyttet de verktøyene de hadde tilgjengelig men heller valgte å bruke de enkleste verktøyene om og om igjen. Det var også

noe overaskende at svært få nevnte noe om fargekoder i deres tilbakemeldinger fra både den fysiske og digitale innsamlingen. Dette gjenspeiler seg også i besvarelsene, hvor det kun var en håndfull av informantene som benyttet seg av fargekodene i deres kode.

Uvanlige og konseptuelle funn: Da de fleste resultatene var relevante for problemstillingen i prosjektet er det ikke mye å rapportere om i denne kategorien. Likevel viste det seg at selv om ikke flytskjemaer var noe som ble utforsket i sammenheng med verktøyene som ble utforsket i litteratursøket eller første fase av prosjektarbeidet viser dette seg å være en form for visualisering helsepersonell liker og kjenner seg igjen i.

4 Diskusjon og refleksjon

Dette kapittelet vil ta for seg diskusjon og refleksjon rundt prosjektets bidrag til feltet i tillegg til implikasjoner for fremtidig arbeid basert på funnene fra datainnsamlingen. Avslutningsvis vil det presenteres steg for veien videre og mer i en konklusjon.

4.1 Vitenskapelig relevans av resultatene

Litteraturgjennomgangen i kapittel 1.2 tar for seg mange aspekter både med visuell programmering, EUD og utfordringer med programmering. I dette underkapittelet vil det vurderes hvorvidt resultatene fra prosjektet kan styrke eller sette spørsmålstegn ved de allerede etablerte teoriene som finnes der ute. All litteraturen som diskuteres her kan leses i kapittelet som tar for seg litteraturgjennomgangen.

Da dette prosjektet ikke direkte omhandlet utvikling og produksjon av et sluttbrukerverktøy er det enkelte teorier som ikke har blitt besvart når det kommer til utfordringene flere forfattere skriver at vi i dag har med visuell programmering og EUD generelt. Likevel er det enkelte ting som skinner gjennom som kan bekrefte flere av disse poengene. En av styrkene til visuell programmering er at det skal være enklere å unngå kodefeil sammenlignet med tradisjonell programmering (Maloney et al., 2010). Noe som kom tydelig frem under datainnsamlingen, var at så lenge informantene hadde materiale å støtte seg på klarte de å forstå koden korrekt i flere tilfeller. Forstå er et nøkkelord her, da det å forstå i denne sammenhengen viser seg å være noe helt annet enn å bruke og skrive kode med syntaksen de har lært. Dette kommer spesielt tydelig frem i oppgaven hvor de skulle kode med penn og papir. Her går det igjen at

syntaksen i språket fort kan bli glemt og informantene forenkler eller skriver ufullstendig kode. Jeg tror derfor at påstanden om at visuell programmering kan hjelpe sluttbrukere å unngå feil stemmer, men at det er to forutsetninger som styrer i hvilken grad disse feilene blir forhindret.

Den første forutsetningen mener jeg blir regulert av hvor mye frihet brukerne får til å lage egne kodekomponenter. Vi kan se i resultatene at så fort vi tar bort biblioteket med kode de har blitt tilbudt i opplæringen og de skal kode på egenhånd at mange av reglene de frem til dette tidspunktet har forholdt seg til raskt blir lagt til side. Ikke bare ble det lagt til side, men informantene valgte å kun benytte seg av de enkleste formene av kode de forstod i enkelte tilfeller. Et eksempel på dette var en diskusjon jeg hadde etter et av de fysiske møtene hvor informanten var usikker på hvordan vi skulle håndtere hendelsesforløpet om lærling var til stede eller ikke. Istedenfor å bruke en if/else statement som var tanken her, foreslo informanten at sykepleier kunne flytte seg til alle de tilgjengelige posisjonene i rommet for å se om lærling var til stede eller ikke. Jeg tror derfor at det er en styrke, om ikke et krav, med et klart definert og utviklet språk som dekker deres behov, slik som et domene spesifikt språk (DSVL). Dette tar oss tilbake til påstanden fra Alan Blackwell om at det ikke finnes noen universell løsning som løser alle sine problemer når det kommer til programmering, men at det heller må ta høyde for problemene det skal løse (Blackwell et al., 2019). Denne påstanden mener jeg det er hold i som vi eksempelvis kan se i at syntaksen og grensesnittet jeg tilbyr i mitt oppgavesett gir de en forståelse så lenge de har læremateriale foran seg, men det er ikke spesifikt nok til at enkelte klarer å benytte seg av dette på egenhånd.

Et annet spennende diskusjonsområdet som nevnes i litteraturen er forskjellen mellom horisontal og vertikal programmering. I de to formene for programmering som blir presentert i oppgavesettene blir begge disse retningene utforsket: Blokkbasert programmering leses vertikalt, mens nodebasert programmering leses både horisontalt og vertikalt. I kapittel 2.7 kan du lese om studiet til Ritschel et al. fra 2020 som så på hvordan industrielt personell kan programmere roboter ved å benytte seg av blokkbasert programmering. Det ble laget tre prototyper til dette prosjektet, hvor alle hadde forskjellige grensesnitt. Selv om ikke dette var målet med prosjektet deres, så observerte de at den horisontale programmeringen viste seg å være enklere å benytte, ikke bare for de som hadde lite erfaring med programmering fra før, men også for de som tidligere hadde skrevet programkode. Etter å ha samlet inn data til dette prosjektet vil jeg si at det også er hold i denne påstanden. Da jeg spurte flere av informantene

om hva de hadde utfordringer med når de satt fast på den blokkbaserte programmeringen var svaret ved to anledninger at det var så ulikt å lese noe på denne måten i forhold til noe de hadde gjort før. Dette dukket aldri opp i den nodebaserte innsamlingen, og jeg tror dette har noe å gjøre med måten vi er vant til å lese tekster på i dag. I en bok står det aldri to-tre ord på en linje før vi går til neste, istedenfor fyller teksten opp den plassen den har horisontalt på siden før det kommer et linjeskift, noe som til dels ligner på den nodebaserte tilnærmingen informantene ble presentert med.

4.2 Implikasjoner for videre design

Som et resultat av forskningen som ble gjennomført i dette prosjektet har vi nå flere bidrag og pekepinner som forhåpentligvis kan hjelpe fremtidige prosjekter å spare både tid og ressurser. Disse bidragene og de veiene som allerede har blitt utforsket vil bli presentert i dette underkapittelet.

Det finnes mange varianter og bruksområder for visuell programmering. Etter litteratursøket satt jeg som nevnt tidligere i oppgaven igjen med to tilnærminger til visuell programmering; blokkbasert og nodebasert. Etter innsamling av resultatene og diskusjonen i foregående kapittel vil jeg si at det også ville vært hensiktsmessig å se nærmere på programmeringsspråk som benytter seg av flytskjemaer i grensesnittene sine dersom man ønsker å utforske mer om visuell programmering rettet mot helsesektoren. De er ikke helt ulike nodebaserte grensesnitt i måten de leses på, og det virker som dette er et format sykepleiere til dels er kjent med. Jeg tror derfor at det er verdt å utforske hvordan man kan låne noen av komponentene fra flytskjemaer slik som de forskjellige formene de bruker på for eksempel valg og input, slik at man kan lage enda mer fleksibel og lesbar kode for denne målgruppen. Å kjøre en tilnærmet lik datainnsamling igjen med det gamle nodebaserte oppgavesettet og en ny hybridvariant med deler av flytskjemaer kan hjelpe oss å komme et steg nærmere å finne en god løsning for helsepersonell. Ingen av informantene i den fysiske innsamlingen hadde noen erfaring med moderne programmering, og det burde tas høyde for at et domene spesifikt språk eller noe lignende semiotikken brukt i Mcreator er helt nødvendig for at verktøyet skal kunne brukes effektivt. Blokkbasert programmering stiller jevnt over svakere på resultatene, og det er trolig mer verdifullt å fokusere på å utvikle et nodebasert grensesnitt med de overnevnte egenskapene. Selv om blokkbasert programmering er den dominerende tilnærmingen brukt for modifisering av spill som nevnt i kapittel 2.2 er dette muligens rettet mot en annen

demografi enn det som er målgruppen her, noe som virker å spille en større rolle enn opprinnelig forventet.

Innledningsvis i oppgaven ble det nevnt at det gjennom oppstartsfasen av prosjektet ikke virket å være noen akademiske prosjekter som har hatt en lignende problemstilling som dette prosjektet. Ved prosjektets avslutning er dette fortsatt tilfellet. Etter å ha snakket med flere praktikanter innenfor fagfeltet syntes jeg det er vanskelig å forstå hvorfor tilsynelatende ingen har gjennomført et prosjekt som dette før. Det virker ikke å være unormalt at simuleringer blir brukt i opplæring for helsepersonell både i høyere utdanning og i større institusjoner som sykehus. Det er flere akademiske artikler som poengterer at virtuelle simuleringer og serious games blir mer og mer brukt for opplæring av helsepersonell(Drummond et al., 2017; Rogers, 2011), men flere av disse fokuserer på effekten og overordnede mål med slike verktøy enn selve oppbygningen. Flere av informantene forklarte hvordan simuleringene fungerte hos dem i dag, i den forbindelse var det to forskjellige former for simuleringer som kom opp for diskusjon.

Den første formen er en tekst basert simulering brukt på høgskolen hvor studentene blir presentert med et tekstlig scenario og skal gjøre valg basert på den informasjonen de får. Disse simuleringene kan håndteres av instruktørene ved høgskolen i et tekstbasert grensesnitt. Slik jeg ble fortalt at det funket så har instruktørene noen muligheter til å gjøre endringer i hendelsesforløpet i simuleringen, men det var hovedsakelig endringer på verdier i forskjellige parametere som gjør at utfallet av valgene til studentene endrer seg. Vi snakket også om en tredimensjonal simulering som ligger mer nært dette prosjektet. Vedkommende som fortalte meg om dette verktøyet sa at de ikke hadde noen måte å manipulere noe i simuleringen selv, men at de heller måtte bestille pakker for forskjellige scenarioer, for så å få det produsert og levert. Med tanke på hvor standarden ligger i dag i den norske helsesektoren og praktikantenes mangel på tilgang til verktøy for manipulasjon av mer komplekse simuleringer syntes jeg disse forklaringene fra informantene understreker viktigheten av videre forskning på dette feltet. Robuste og fleksible sluttbrukerverktøy er mangelvare i markedet, og jeg tror det delvis skyldes at vi i dag ikke vet helt hvordan vi skal håndtere de forskjellige aspektene med en simulering i og med at målgruppene har såpass forskjellige preferanser.

Direkte manipulasjon skal ifølge Ben Shneiderman føre til bedre opplevelser hos sluttbrukerne. Noe jeg merket godt under den fysiske innsamlingen men som også gjenspeilet

seg i de digitale besvarelsene var at brukerne til dels savnet en form for respons. Istedenfor at de fikk en respons fra systemene når de besvarte oppgavene spurte de heller meg om de hadde gjort det riktig eller feil under intervjuene. På oppgavene med svaralternativer er dette enkelt nok, enten så hadde du riktig eller så hadde du feil, men dette er ikke nødvendigvis tilfellet på oppgaven hvor de skal kode selv. Som nevnt i kapittel 2.2.2 som tar for seg Game Master Mode, så er dette et av de få eksemplene som faktisk benytter seg av direkte manipulasjon i sin løsning ved at brukerne til enhver tid ser hvordan endringene deres påvirker flyten i spillet samtidig som alt er reversibelt.

Tatt i betraktning at brukerne sliter såpass med å se resultatene av det de koder og i flere tilfeller faktisk mangler flere av elementene de blir bedt om å kode mener jeg man burde lage et grensesnitt som støtter noen av disse egenskapene som Shneiderman nevner som grunnlaget for direkte manipulasjon. Et verktøy som til en grad innehar noen av disse egenskapene i et programmeringsgrensesnitt er Bolt, som lar oss se hvordan logikken flyter gjennom nodene ved å bruke grafiske elementer. I tillegg til dette kan man gjøre endringer i koden sin i sanntid, slik at man kan se hvordan endringene påvirker spillet mens man programmerer istedenfor at man må starte og stoppe spillet hver gang man gjør en endring.

4.3 Konklusjon

Etter å ha sett nærmere på to tilnærminger til visuell programmering står jeg nå igjen med et svar på forskningsspørsmålet til dette masterprosjektet. Forskningsspørsmålet vi har besvart er som følger: *Hvilken av de to mest populære visuelle programmerings grensesnittene (blokkbasert og nodebasert) balanserer best kompleksitet og brukervennlighet for instruktører som har i oppgave å lage scenariobaserte virtuelle simuleringer som brukes til opplæring av helsepersonell?*

Innledningsvis var problemstillingen mer åpen, hvor målet var å se på alle former for visuell programmering. Tidlig fant jeg likevel ut at det var to dominerende former for visuell programmering, henholdsvis nodebasert- og blokkbasert programmering, og tilpasset problemstillingen deretter. For å besvare forskningsspørsmålet ble prosjektarbeidet delt opp i fire faser: innsikt, idéutvikling, utvikling og testing. Det finnes mange verktøy som benytter seg av visuell programmering i forskjellige former, og det var derfor først naturlig å se på verktøyene som blir brukt i dag. Det var ikke tilfeldig hvilke av disse som ble valgt, og jeg

valgte å redusere utvalget til verktøy som blir brukt til manipulering og modifikasjon av dataspill da dette er nærliggende det overordnede målet med dette prosjektet. Etter å ha testet de mest populære verktøyene som ble brukt i dag ble første iterasjon av et programmeringsspråk utviklet. Dette språket håndterte kun naturlig språk og ville derfor ikke være omfattende nok til å kunne svare på forskningsspørsmålet. I fase 3 ble derfor andre iterasjon av språket bygget laget, og tok grunnlag i det jeg hadde lært i forrige fase, dette resulterte i to oppgavesett rettet mot hver av tilnærmingene til visuell programmering.

For å besvare forskningsspørsmålet har det blitt gjennomført to forskjellige kvalitative datainnsamlinger som har gitt meg den informasjonen jeg trenger for å føle meg trygg på min konklusjon. Den ene innsamlingen var et digitalt spørreskjema som inneholdt et sett med oppgaver informantene ble bedt om å løse. De hadde også muligheten til å skrive ned tanker og tilbakemeldinger underveis i oppgavesettet. Denne innsamlingen ga meg et overblikk over hvor svakhetene i både programmeringen i tillegg til mulige svakheter ved informantenes forståelse av programmering og den logiske tenkningen som følger med det å skrive programkode. Etter den digitale datainnsamlingen ble det også gjennomført fysiske intervjuer hvor jeg fikk muligheten til å sette meg ned med både de som praktiserer eller har praktisert sykepleie i tillegg til noen av de som til daglig håndterer simuleringene på Høgskolen i Østfolds avdeling for helse og velferd i Fredrikstad. Under disse intervjuene gikk vi gjennom de samme oppgavesettene, men jeg fikk også høre hvordan de tenkte da de løste oppgavene. Da dette var erfarne informanter innenfor sykepleien fikk jeg også en del informasjon om hvordan dette gjøres i dag, og det var flere som brukte erfaringer fra arbeidslivet til å sammenligne min løsning med deres simuleringer som de bruker jevnlig i jobben sin.
Basert på disse to innsamlingene og litteratursøket konkluderer jeg med at svaret på forskningsspørsmålet er at nodebasert programmering er det beste alternativet av de to jeg har utforsket for vår målgruppe. Oppgavesettet jeg utviklet er likevel ikke et perfekt eksempel på nodebasert programmering da dette er fiktivt og har redusert funksjonalitet. Resultatene var fortsatt overbevisende nok til at jeg føler meg trygg på denne konklusjonen.

Selv om blokkbasert programmering virker å være den mest utbredte formen for visuell programmering i sluttbrukerverktøy for modifisering av spill viste det seg at det ikke var det som passet best for målgruppen til dette prosjektet. Da dette prosjektet ikke har produsert noe produkt men heller har generert kunnskap, så er det trolig flere aspekter, styrker og svakheter

med denne formen for programmering man kun vil finne dersom man tar det et steg videre med en fungerende prototype.

4.3.1 Videre arbeid

Som nevnt over er det mye mer man må finne ut av for å finne den optimale programmeringsløsningen som balanserer kompleksitet og fleksibilitet for helsepersonell. Da dette prosjektet har fokusert mest på å finne ut hvilken tilnærming man burde gå for, så er et naturlig neste steg å lage et mer komplekst og interaktivt grensesnitt som benytter nodebasert programmering. Ved å ta inspirasjon fra punktene nevnt i kapittel 4.2, da spesielt direkte manipulasjon, mener jeg man kan få verdifull informasjon som kan hjelpe oss å forme et grensesnitt som gir målgruppen vår de verktøyene de trenger til å manipulere og lage scenariobaserte opplæringsverktøy. Dersom dette ikke er oppnåelig ved å kun benytte seg av noder tror jeg også at å flette inn elementer fra flytskjemaer kan være hensiktsmessig da dette virker som å være et format helsepersonell til dels er kjent med og kanskje kan gjøre programmering litt mer tilgjengelig.

Referanser

Barricelli, B. R., Cassano, F., Fogli, D., & Piccinno, A. (2019). End-user development, end-user programming and end-user software engineering: A systematic mapping study. *Journal of Systems and Software*, *149*, 101–137. https://doi.org/10.1016/j.jss.2018.11.041

Blackwell, A. F. (2002). First steps in programming: A rationale for attention investment models. *Proceedings IEEE 2002 Symposia on Human Centric Computing Languages and Environments*, 2–10. https://doi.org/10.1109/HCC.2002.1046334

Blackwell, A. F., Petre, M., & Church, L. (2019). Fifty years of the psychology of programming. *International Journal of Human-Computer Studies*, *131*, 52–63. https://doi.org/10.1016/j.ijhcs.2019.06.009

Chu, E., & Zaman, L. (2021). Exploring alternatives with Unreal Engine's Blueprints Visual Scripting System. *Entertainment Computing*, *36*, 100388. https://doi.org/10.1016/j.entcom.2020.100388

Costabile, M. F., Fogli, D., Mussio, P., & Piccinno, A. (2007). Visual Interactive Systems for End-User Development: A Model-Based Design Methodology. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, *37*(6), 1029–1046. https://doi.org/10.1109/TSMCA.2007.904776

Costabile, M. F., Mussio, P., Parasiliti Provenza, L., & Piccinno, A. (2008). End users as unwitting software developers. *Proceedings of the 4th International Workshop on End-User Software Engineering - WEUSE '08*, 6–10. https://doi.org/10.1145/1370847.1370849

Costabile, M. F., Piccinno, A., Fogli, D., & Marcante, A. (2006). Supporting interaction and co-evolution of users and systems. *Proceedings of the Working Conference on Advanced Visual Interfaces*, 143–150. https://doi.org/10.1145/1133265.1133294

Costabile, M.-F., Fogli, D., Letondal, C., Mussio, P., & Piccinno°, A. (2003, June). Domain-Expert Users and their Needs of Software Development. *HCI 2003 End User Development Session*. https://hal.archives-ouvertes.fr/hal-01299738

Creswell, J. W., & Creswell, J. D. (2018). *Research design: Qualitative, quantitative & mixed methods approaches* (5th edition). Sage.

Cunningham, K., Ericson, B. J., Agrawal Bejarano, R., & Guzdial, M. (2021). Avoiding the Turing Tarpit: Learning Conversational Programming by Starting from Code’s Purpose. In *Proceedings of the 2021 CHI Conference on Human Factors in Computing Systems* (pp. 1–15). Association for Computing Machinery. https://doi.org/10.1145/3411764.3445571

Drummond, D., Hadchouel, A., & Tesnière, A. (2017). Serious games for health: Three steps forwards. *Advances in Simulation*, *2*(1), 3. https://doi.org/10.1186/s41077-017-0036-3

Fischer, G. (2009). End-User Development and Meta-design: Foundations for Cultures of Participation. In V. Pipek, M. B. Rosson, B. de Ruyter, & V. Wulf (Eds.), *End-User Development* (Vol. 5435, pp. 3–14). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-00427-8_1

Fischer, G., & Giaccardi, E. (2006). Meta-design: A Framework for the Future of End-User Development. In H. Lieberman, F. Paternò, & V. Wulf (Eds.), *End User Development* (Vol. 9, pp. 427–457). Springer Netherlands. https://doi.org/10.1007/1-4020-5386-X_19

Fischer, G., Giaccardi, E., Ye, Y., Sutcliffe, A. G., & Mehandjiev, N. (2004). Meta-design: A manifesto for end-user development. *Communications of the ACM*, *47*(9), 33–37. https://doi.org/10.1145/1015864.1015884

Fischer, G., Nakakoji, K., & Ye, Y. (2009). Metadesign: Guidelines for Supporting Domain Experts in Software Development. *IEEE Software*, *26*(5), 37–44. https://doi.org/10.1109/MS.2009.134

Foronda, C., MacWilliams, B., & McArthur, E. (2016). Interprofessional communication in healthcare: An integrative review. *Nurse Education in Practice*, *19*, 36–40. https://doi.org/10.1016/j.nepr.2016.04.005

Hutchins, E. L., Hollan, J. D., & Norman, D. A. (1985). Direct Manipulation Interfaces. *Human–Computer Interaction*, *1*(4), 311–338. https://doi.org/10.1207/s15327051hci0104_2

King, D., Tee, S., Falconer, L., Angell, C., Holley, D., & Mills, A. (2018). Virtual health education: Scaling practice to transform student learning. *Nurse Education Today*, *71*, 7–9. https://doi.org/10.1016/j.nedt.2018.08.002

Ko, A. J., Abraham, R., Beckwith, L., Blackwell, A., Burnett, M., Erwig, M., Scaffidi, C., Lawrance, J., Lieberman, H., Myers, B., Rosson, M. B., Rothermel, G., Shaw, M., & Wiedenbeck, S. (2011). The state of the art in end-user software engineering. *ACM Computing Surveys*, *43*(3), 21:1-21:44. https://doi.org/10.1145/1922649.1922658

Maloney, J., Resnick, M., Rusk, N., Silverman, B., & Eastmond, E. (2010). The Scratch Programming Language and Environment. *ACM Transactions on Computing Education*, *10*(4), 16:1-16:15. https://doi.org/10.1145/1868358.1868363

Marchiori, E. J., del Blanco, Á., Torrente, J., Martinez-Ortiz, I., & Fernández-Manjón, B. (2011). A visual language for the creation of narrative educational games. *Journal of Visual Languages & Computing*, *22*(6), 443–452. https://doi.org/10.1016/j.jvlc.2011.09.001

Mason, D., & Dave, K. (2017). Block-based versus flow-based programming for naive programmers. *2017 IEEE Blocks and Beyond Workshop (B B)*, 25–28. https://doi.org/10.1109/BLOCKS.2017.8120405

Menestrina, Z., & De Angeli, A. (2017). End-User Development for Serious Games. In F. Paternò & V. Wulf (Eds.), *New Perspectives in End-User Development* (pp. 359–383). Springer International Publishing. https://doi.org/10.1007/978-3-319-60291-2_14

Menestrina, Z., De Angeli, A., & Busetta, P. (2014). APE: End User Development for Emergency Management Training. *2014 6th International Conference on Games and Virtual Worlds for Serious Applications (VS-GAMES)*, 1–4. https://doi.org/10.1109/VS-Games.2014.7012030

Morrison, C., Villar, N., Thieme, A., Ashktorab, Z., Taysom, E., Salandin, O., Cletheroe, D., Saul, G., Blackwell, A. F., Edge, D., Grayson, M., & Zhang, H. (2020). Torino: A Tangible Programming Language Inclusive of Children with Visual Disabilities. *Human–Computer Interaction*, *35*(3), 191–239. https://doi.org/10.1080/07370024.2018.1512413

Mørch, A. (1997). Three levels of end-user tailoring: Customization, integration, and extension. *Computers and design in context,* 51-76.

Repenning, A. (2000). *AgentSheets: An Interactive Simulation Environment with End-User Programmable Agents*. Undefined. /paper/AgentSheets%3A-an-Interactive-Simulation-Environment-Repenning/49c40ffc08045bcfd5107f3de7aa81491c44d4e8

Repenning, A., & Sumner, T. (1995). Agentsheets: A medium for creating domain-oriented visual languages. *Computer*, *28*(3), 17–25. https://doi.org/10.1109/2.366152

Ritschel, N., Kovalenko, V., Holmes, R., Garcia, R., & Shepherd, D. C. (2020). Comparing Block-based Programming Models for Two-armed Robots. *IEEE Transactions on Software Engineering*, 1–1. https://doi.org/10.1109/TSE.2020.3027255

Rogers, L. (2011). Developing simulations in multi-user virtual environments to enhance healthcare education: Developing simulations to enhance healthcare education. *British Journal of Educational Technology*, *42*(4), 608–615. https://doi.org/10.1111/j.1467-8535.2010.01057.x

Sharifzadeh, N., Kharrazi, H., Nazari, E., Tabesh, H., Edalati Khodabandeh, M., Heidari, S., & Tara, M. (2020). Health Education Serious Games Targeting Health Care Providers, Patients, and Public Health Users: Scoping Review. *JMIR Serious Games*, *8*(1), e13459. https://doi.org/10.2196/13459

Shneiderman, B. (1997). Direct manipulation for comprehensible, predictable and controllable user interfaces. *Proceedings of the 2nd International Conference on Intelligent User Interfaces*, 33–39. https://doi.org/10.1145/238218.238281

Sprinkle, J., & Karsai, G. (2004). A domain-specific visual language for domain model evolution. *Journal of Visual Languages & Computing*, *15*(3), 291–307. https://doi.org/10.1016/j.jvlc.2004.01.006

Tanaka-Ishii, K. (2010). *Semiotics of Programming*. Cambridge University Press.

Weintrop, D., & Wilensky, U. (2017). Comparing Block-Based and Text-Based Programming in High School Computer Science Classrooms. *ACM Transactions on Computing Education*, *18*(1), 3:1-3:25. https://doi.org/10.1145/3089799

Westera, W. (2017). How people learn while playing serious games: A computational modelling approach. *Journal of Computational Science*, *18*, 32–45. https://doi.org/10.1016/j.jocs.2016.12.002