Forord

Dette kompendiet beskriver, med noen få eksempler, torgmodellen for utvikling av pedagogisk programvare. Modellen er svært uformell, og må mer ses på som en oversikt over steg man bør gå gjennom når man skal utvikle et program. Den tilbyr en del mentale verktøy man kan benytte for å sikre seg at man lager et interaktivt og engasjerende program. I tillegg legger den stor vekt på å sørge for at man fokuserer på de rette tingene, de elementene i designet som skal få eleven til å fokusere på det tema som skal læres, og som skal være med på å bygge opp en mental modell elevene kan benytte for å forstå problemstillingen.

Innledning

Datamaskiner har et stort potensiale for bruk i skolen. Det er mye forskjellig programvare som kan utnyttes, både generelle verktøypakker (som iWork og MS Office) og programvare skrevet spesielt for skoleverket. Ofte kan man se pedagogisk programvare delt inn i gruppene øvingsprogrammer og datapedagogiske verktøyprogram. Dette er kanskje misvisende siden alle programmene kan oppfattes som datapedagogiske verktøy, men det kan være lurt å skille på denne måten. Fokus i de senere årene er mer og mer vendt mot verktøysaspektet av datateknologien

I den nevnte klassifiseringen tenker man seg verktøysprogrammene inndelt i emneorienterte- og kontekstfrie verktøy. Felles for disse er at de:

- inneholder ingen læringsstrategi
- stiller til rådighet et sett verktøy egnet til å løse faglige problemer utenfor programmet

I tillegg gjelder det for de emneorienterte verktøyene at den faglige sammenhengen for verktøyet allerede er fastlagt, mens de kontekstfrie verktøyene er åpne verktøy der brukssammenhengen ikke er definert på forhånd.

For øvingsprogrammene er dette noe anderledes. Disse karakteriseres ved at:

- de inneholder identifiserbare fagmål
- inneholdet er begrenset til et fast stoffområde
- de hjelper i å trenebegrepsforståelse og ferdigheter
- de gir muligheter til å øve regler og/eller faktakunnskaper

Øvigsprogrammene deles igjen inn i 3 grupper: repeterende, eksperimenterende og adaptive.

Repeterende øvingsprogram:

- Framstiller oppgaver i en bestemt rekkefølge
- Mottar og aksepterer/avviser svar etter et fastlagt mønster.

Eksperimenterende øvingsprogram:

Skaffer nye erfaringer

- Læring av begrep, oppdagelse av prinsipp, tilegnelse av ferdighet

Adaptive øvingsprogram

- Tilpasser seg brukeren etter det bilde programmet dannert seg av ham
  - vanskegrad, dialogform, faghjelp
  - i noen tilfeller også læringsstil.

Pedagogisk programvare

På 1970-tallet hadde man ideen om "Courseware". Til hvert emne skulle det finnes en læremodul man skulle gjennom for å lære den spesielle delen av pensum. Når man hadde vært gjennom denne læremodulen skulle man testes for å se om man kunne innholdet i modulen. Hvis man besto testen gikk man videre til neste modul, osv. til man var ferdig med pensum. Da skulle man kunne stoffet. Elevene ble bundet i en fast rute med fastlagt progresjon og med svært liten frihet for elevene. Det ble utviklet ganske mye programvare etter denne tankegangen, ofte med systemet PLATO.

På 1980-tallet ble denne tankegangen modifisert og man fikk idéen om "lessonware". Dette er programvare som skal være langt friere og som skal være tilpasset bruk i vanlig undervisning. Mens man med "courseware" kunne tenke seg store datalaber med mange elever som satt og jobbet med programvare, skiftet fokus fra store datalaber til heller å se på dataprogrammer som noe som skulle inngå som en naturlig del av et læringsmilhjø i klasserommet, der bare en del av elevene skulle jobbe med programvaren ved behov, og ikke alle skulle jobbe med programvaren uansett behov eller ikke.

"Lessonware" blir beskrevet som åpen programvare som skal:

- brukes for å løse spesifikke pedagogiske problemer
- være tverrfaglige
- være velegnet til bruk i prosjekter

Selv om "lessonware" skal være fri og åpen, kan man tenke seg den organisert på 2 forskjelllige måter. Man kan bruke en "museumsmetafor" for å beskrive forskjellen på disse to. Et museum kan organiseres på (i hvertfall) to måter.

På Ringve museum i Trondheim organiseres man i grupper og får tildelt en guide som tar en gjennom museet og sørger for at alle de viktigste tingene blir vist fram. Dette er en "ledet oppdagelsestur" gjennom museet, der man passer på at deltageren får muligheten til å få med seg alt det viktigste. Brukeren er ikke fri til å gjøre hva man vil, men man har en mulighet til å lede deltageren innom de sentrale elementene som det er viktig å ta med seg.

På et større kunstmuseum, f.eks. Uffizi i Firenze eller Hermitagen i St. Petersburg, kjøper man billett og slippes inn slik at man kan vandre rundt å se det man vil. Dermed er man fri til å gjøre hva man vil, i den rekkefølgen man

vil, men med det problemet at man kan komme ut av museet uten at man har fått med seg alt. Imidlertid er det kanskje en større mulighet for at man besøker museet flere ganger.

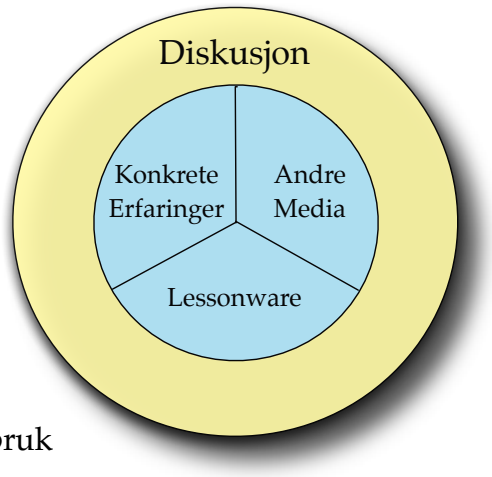
Den metoden som beskrives i kompendiet er lagd for å lage denne typen programvare ("lessonware"). Utgangspunktet er den frie typen (eksemplifisert gjennom "Hermitage"), men den egner seg også til å lage "Ringve museum"-type programvare.

Lessonware

Lessonware kan kort beskrives som "et relativt kort program som komplementerer, forsterker, evt. enkelte ganger erstatter en skoletime". God lessonware:

- fokuserer mot det vanskelige
- aktiviserer eleven (både fysisk og mentalt)
- går fra enkelt til komplekst
- er klar og forståelig
- involverer mer enn en sans
- er fargerik
- er interessant og spennende
- krever deltagelse
- er relevant

Dvs. den er bygd på de samme prinsippene som en god skoletime. Lessonware skal lages som noe som hører naturlig med i en læringssituasjon (om det er i en skoletime eller ikke).



Figuren er ment å illustrere de elementene en god skoletime består/kan bestå av. Gjennom diskusjon og sosial interaksjon integrerer man konkrete erfaringer, erfaringer gjort gjennom bruk

av programvare, og informasjon man har fått fra andre media til en komplett læringssituasjon.

Hvor finner man så de gode idéene? En god kilde er de konkrete erfaringene man har gjort seg som elev eller lærer - hvilke emner har elevene vanskelig for å forstå, og hva består problemene i? Andre kilder til idéer kan være konkrete erfaringer fra dagliglivet samt eksisterende fagplaner og lærebøker. Ofte er elevene umotiverte fordi de ikke ser nytten i det de skal lære, og hvis man finner en god illustrasjon fra dagliglivet på et prinsipp man skal lære på skolen, kan det være en god idé for et program.

I mange lærebøker finnes det tema som er dårlig behandlet. Dette kan skyldes at hjelpemiddelet (lærebok) ikke egner seg godt for behandling av emnet (det er ikke sikkert at emnet egner seg til å bli beskrevet i et statisk medium som en bok), eller kanskje forfatteren rett og slett ikke har gjort en god nok jobb?

I andre tilfeller er det ting man gjerne ville undervist i, men som ikke står i fagplanen. En del emner kan være utelatt av fagplanen fordi det ikke finnes verktøy slik at emnene kan behandles godt nok. Det som undervises i skolen vil være influert av mange ting, bl.a. hvilke verktøy om er tilgjengelige for lærer og elev. Derfor bør man kanskje ikke være for bundet av allerede eksisterende fagplaner når man skal klekke ut idéer til programvare.

En annen kilde for idéer kan være de problemene som har sitt utspring i differensiering av fagstoff for forskjellige elevgrupper. Dette kan være differensiering av fagstoffet basert på elever med forskjellige fysiske, eller det kan være en differensiering av tempo eller presentasjonsstil.

Momentliste:

- hvilke emner er vanskelig å forstå?
- hvor finnes det mye kjedelig repetisjon og kopiering?
- hvor finnes det motoriske problemer?
- hvor trengs det dyrt, evt. utilgjengelig utstyr?
- hvor er det fare assosiert med elevaktivitet?
- hvilke emner er overflatisk eller dårlig presentert i lærebøker?
- hvilke emner trenger mer tid?
- hvilke emner trenger differensiert undervisning?

Husk også at et program kan brukes på mange måter i en skoletime:

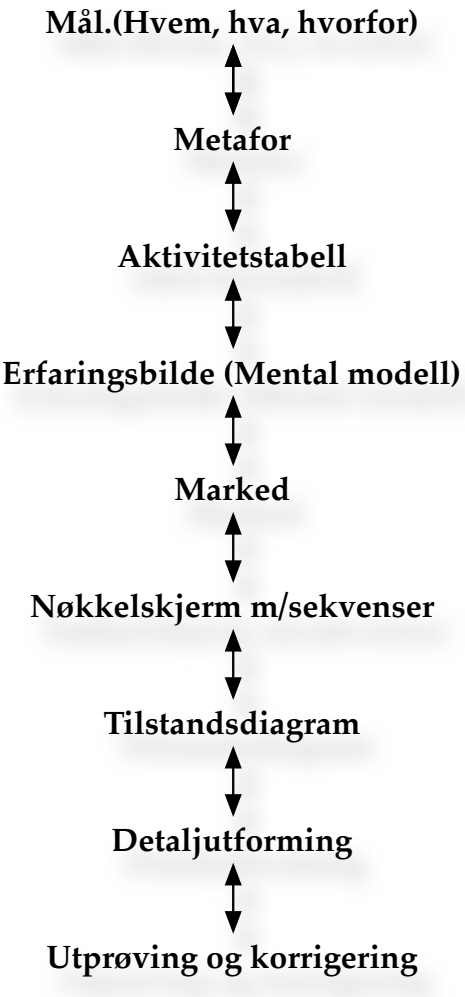
- i gruppearbeid
- som demonstrasjon
- å lære et vanskelig begrep
- som individuell øvelse
- som et verktøy
- som en introduksjon
- som repetisjon
- som forberedelse før en prøve
- som en introduksjon
- som ekstra eksempler for noen elever.

Elever har forskjellig bakgrunn og måter å lære på. Dette tilsier at man må kunne jobbe i sitt eget tempo og følge sin egen sti gjennom programmet, noe som igjen tilsier at man må ha en åpen og interaktiv programvare. Samtidig må det gi hint og lede elevene til riktige konklusjoner.

Man lærer på forskjellige måter

- Auditive (få ting forklart muntlig)
- Visuelle (se ting)
- Utforskende (prøve på egen hånd)
- Følge oppskrifter

Oversikt over modellen



Modellen oppfordrer til å gjøre endringer underveis. Tro ikke at et tema er utdebatert. Man vil ofte gjøre endringer som vil føre til forandringer i tidligere deler av designet. Man må alltid være forberedt på å gjenoppta diskusjoner som man kanskje har regnet med å være ferdig med.

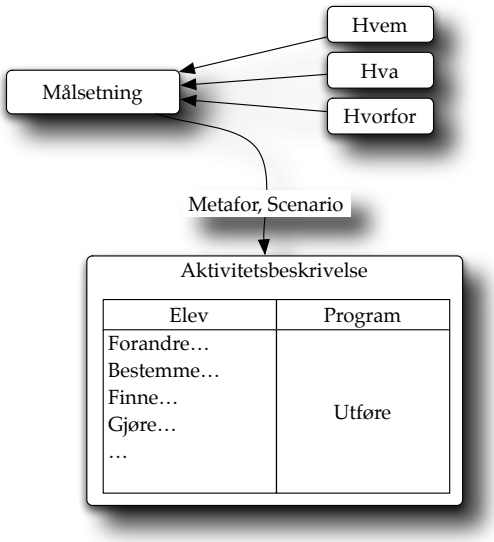
Figuren til venstre viser denne iterative arbeidsmåten. Ethvert punkt er alltid debaterbart og mulig å endre til alle tider. Man bør imidlertid ha svært gode grunner for å gå tilbake og endre tidligere vedtatte punkter.

Man begynner med å spesifisere målsettingen for programmet. Denne skal være mest mulig konkret, og skal inneholde en beskrivelse av hva programmet skal gjøre, hva eleven skal lære, hvilken målgruppe programmet sikter etter, samt en argumentasjon om hvorfor man ønsker å lage et dataprogram for dette.

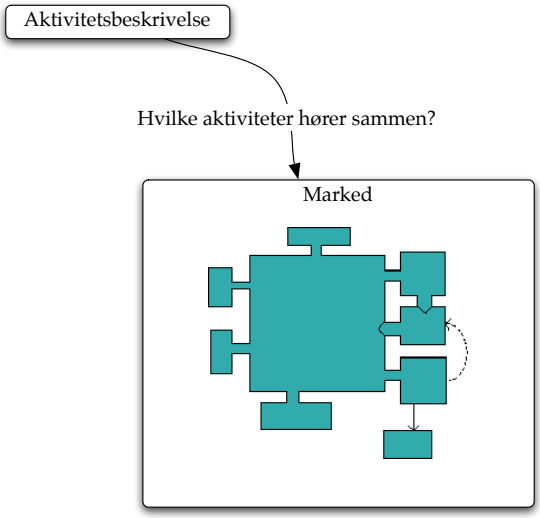
Deretter må man prøve å bestemme en metafor for programmet. I dette ligger å bestemme i hvilke omgivelser eleven skal befinne seg når programmet er aktivt.

Aktivitetstabellen skal gi en oversikt over alle aktivitetene eleven skal utføre som bruker av programmet. Det er et verktøy for å sørge for at programmet aktiviserer eleven i stor nok grad. Disse aktivitetene kan være både fysiske aktiviteter (som at eleven skal flytte noe på skjermen eller lignende) og mentale aktiviteter (som at han/hun skal bestemme seg for hva som er viktig/riktig å gjøre i en gitt situasjon). Tanketom, fysisk aktivitet fremmer ikke læring - man må sørge for at eleven blir stimulert mentalt.

Hvilke aktiviteter det er riktig å gjøre vil være avhengig av hvilken metafor som er valgt og hva slags scenarioer man ser for seg at programmet vil bli brukt i. Derfor er det viktig å ha gjennomdrøftet dette før man lager aktivitetstabellen.

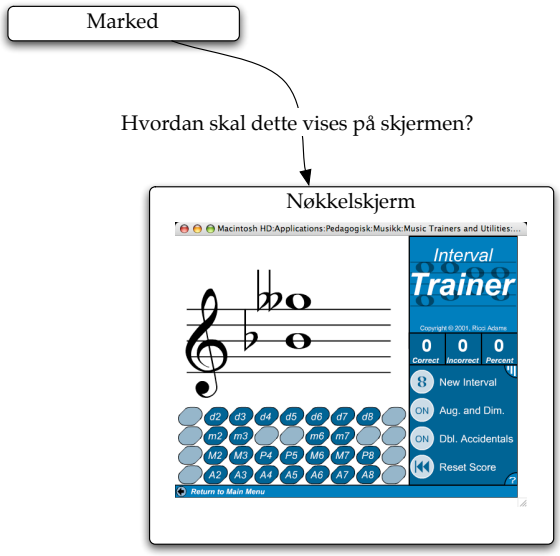


Gjennom metafor- og aktivitetsbeskrivelsen, vil det dukke frem de elementene i programmet som er viktig for at det skal skje læring. Det er disse elementene som kalles "erfaringsbildet" (etter det mentale bildet eleven bygger opp om

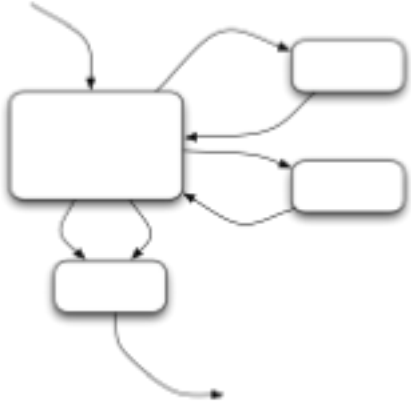


problemområdet. Ofte kalles dette også for en "mental modell"). Hvordan disse elementene henger sammen og skal kunne manipuleres, beskriver man så i et markedsdiagram. Dette diagrammet skal være en enkel skisse over de aktivitetene eleven skal utføre og hvilke sammenhenger det er mellom dem. Det er når eleven utfører disse aktivitetene læringen skjer, og det er derfor viktig å få en oversikt over hvordan disse henger sammen. Slike sammenhenger er vanskelige å beskrive nøyaktig på papiret, og derfor benytter man denne teknikken for å gi et visuelt inntrykk av sammenhengene i programmet.

Når vi har en slik sammenheng klart for oss kan vi begynne å designe hvordan dette skal se ut på skjermen. Det er først nå vi skal begynne å fundere på hvor ting skal plasseres, hvilke "widgets" vi skal bruke, hvilke farger man skal bruke på de forskjellige elementene, hvilke tiltak vi skal ta ibruk for å tiltrekke oppmerksomhet dit det er viktig å se osv… Og det er først nå vi begynner å designe interaksjonen i programmet.



Når vi har dette klart for oss, må vi beskrive hvordan de forskjellige aktivitetene fører til endringer i skjermbildenes tilstand. Dette gjøres i et tilstandsdiagram. Alle aktiviteter en elev gjør skal ha en pil som peker ut av den tilstanden man står i. Denne pilen vil å enten føre til en annen tilstand, eller den vil peke tilbake til den opprinnelige tilstanden, med en beskrivelse av hvordan denne tilstanden endrer seg som en funksjon av denne aktiviteten.



Eksempel

Læreren erfarer problemer med å lære bort lengde- og breddegrader og deres sammenheng med posisjon og plassering på jordkloden. Hun bestemmer seg for å lage et program for å hjelpe til med dette.

Elevene går i 5. klasse, og er dermed 11-12 år gamle. På dette klassetrinnet vil man ha stor variasjon i elevenes modenhet og abstraksjonsevne. Det vil derfor være et generelt problem for elevene å koble et punkt på jordkloden med et koordinatpunkt. Dette blir ikke enklere av at man bruker begrep som grader, minutter og sekunder for å beskrive abscisse og ordinat i punktet.

Målsetningen blir derfor

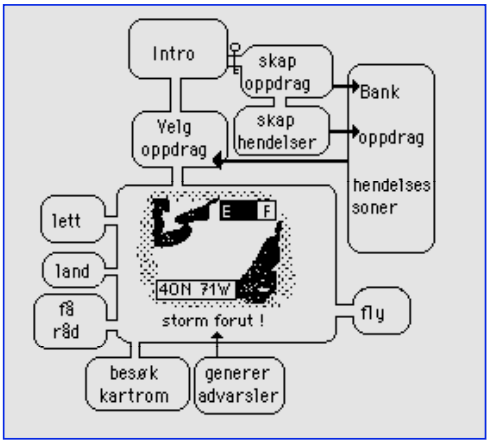
- Elever i 5. klasse
- Forstå begrepene lengde- og breddegrad
- Koble begrepene med avstand
- Plassere steder/land på kloden med oppgitt lengde- og breddegrad.

Siden dette er elever som vil kunne motiveres av spill, velger man **redningsaksjoner** der eleven skal spille rollen som en flyger som får i oppdrag å fly nødsendinger til forskjellige steder på jorden. For å finne fram må han/hun finne frem basert på posisjon oppgitt som lengde- og breddegrad. Med denne metaforen i bakhodet kommer man fram til følgende aktivitetstabell:

Det er viktig at aktivitetene på elevens side virkelig er aktiviteter som krever en aktiv, gjennomtenkt handling fra elevens side. Programmet skal, stort sett, være en passiv deltager som bare reagerer på aktiviteter eleven står for.

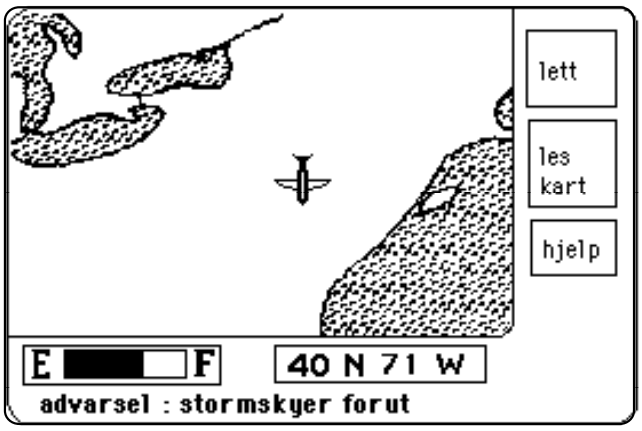
Elevens oppgaver	Programmets oppgaver
lette	gi advarsler
lande	beregne lengde- og breddegrader
fly	bruke opp bensin
sjekke kart	gi tilbakemeldinger på elevens valg
få råd	…
velge oppdrag	…
…	…
…	…

Gjennom arbeidet meddenne aktivitetstabellen kommer man fram til hvilke elementer det er viktig å integrere i markedsdiagrammet. De elementene det blir viktig å få med blir et kart, en markør som viser hvor man befinner seg (et fly), hvilke koordinater det punktet man befinner seg i, hvilke koordinater man skal til, en indikator som viser hvor bra man gjør det (bensinmåler?), en indikator på om man beveger seg i rett retning osv… Det er disse elementene som utgjør **erfaringsbildet**, de elementene man skal bruke for å bygge opp en forståelse for problemet.



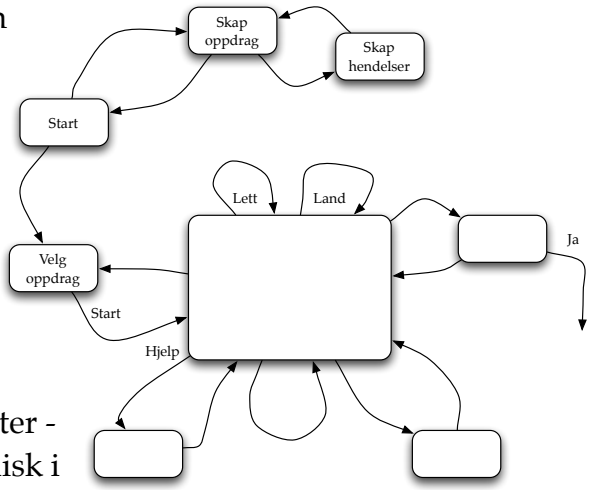
Dette kan f.eks. føre til markedet som er vist i figuren til høyre. De viktige elementene fra erfaringsbildet er plassert på den sentrale delen av torget, mens aktivitetene i aktivitetstabellen finner vi igjen i bodene rundt torgplassen. Selve skjermbildet har vi enda ikke begynt på.

Hvordan dette vil se ut er avhengig bl.a. av de standarder som gjelder for det systemet vi designer for. Et eksempel på et mulig skjermbilde for dette programmet kan være:



Sentralt i skjermbildet er kartet over området eleven flyr over. I tillegg ser man bensinmåleren, posisjon og en advarsel. Disse er alle sentrale for læringen, altså en del av **erfaringsbildet**. Nå er dette plassert inn på skjermen. I tillegg har man med noen enkle knapper som er med til å hjelpe å bruke programmet. Disse er ikke sentrale i læringen, men er kontroller man trenger å ha med. Det er mange ting man må diskutere i dette skjermbildet. Hva skjer når man ber om hjelp? Hva skjer når man konsulterer kartrommet? Skal flyet stå stille og karte rulle, eller skal flyet bevege seg på kartet? Hva skjer i så fall når flyet når ytterkanten av skjermbildet? Hvordan skal man navigere flyet? Mus? Piltaster? noe annet? Etc…

Til slutt beskrives elevens interaksjon med programmet gjennom et tilstandsdiagram som kan se noe slik ut (ikke komplett):



Dette tilstandsdiagrammet modellerer grensesnittet mot bruker, altså brukerdialogen, for å avsløre inkonsistens i denne. Generelt kan man si at tilstandsdiagrammet tjener to hensikter - å tvinge designeren til å tenke metodisk i spesifiseringen av skjermdialogen, og å dokumentere denne på en mest mulig konsis måte til programmereren slik at problemer kan avsløres på et tidlig tidspunkt. De andre punktene i designmetoden er først og fremst tenkt som hjelpemidler for designeren i designfasen - "knagger" for å henge idéer på og viktige verktøy for å sikre at man tenker grundig gjennom de viktigste elementene. Tilstandsdiagrammet kommuniserer også mot programmereren.

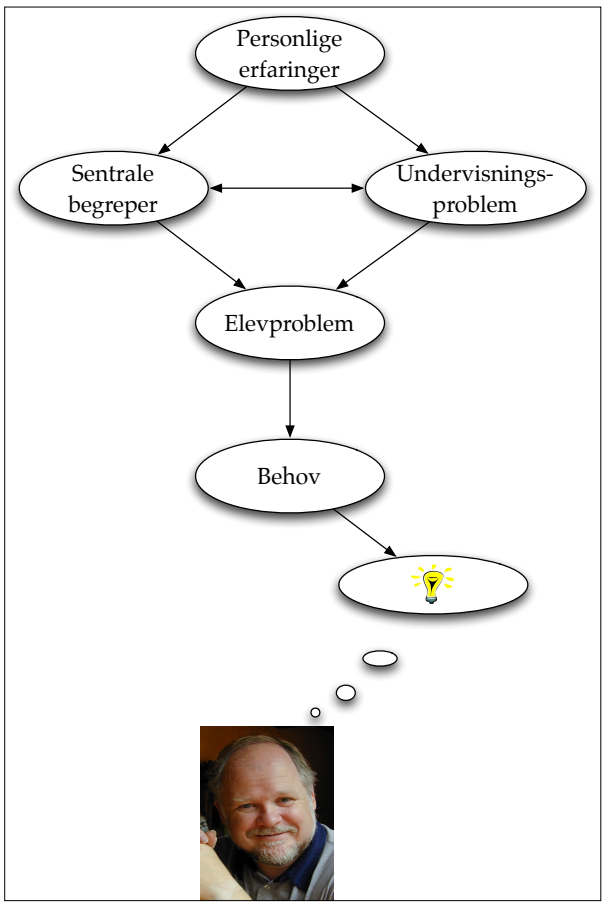
Fallgruver

Det er mange som gjerne vil finne opp hjulet (for ikke å si kruttet) på nytt. Det er svært mange "hjul" som allerede er funnet opp. Typiske eksempler på dette er programvare som databaser, regneark og tekstbehandlere. Det designverktøyet som beskrives i dette kompendiet er ikke spesielt godt egnet til å designe slike generelle verktøy. Hvis det skal være aktuelt å utvikle nye standardprogram som de som er nevnt over for skolebruk, må det dreie seg om program som er spesielt tilpasset en spesiell brukergruppe, f.eks. elevgruper med spesielle behov.

Man bør heller ikke bruke tid på å designe programvare for funksjoner der tradisjonelle metoder fungerer godt. I mange sammenhenger fungerer papir og blyant utmerket. Mange vil gjerne utvikle spill for å f.eks. forbedre regnekunnskapene til eleven, men ofte kan man heller la elevene spille slike spill mot/med hverandre i timene med tradisjonelle verktøy.

En felle det er lett å falle i er å bruke veldig mye tekst. Tekst er en nødvendighet, men man skal være forsiktig med hvordan man bruker det. hvis skjermen fylles opp med tekst, vil programmet fungere nesten som en automatisk bladvender, og til dette egner datamaskinen seg ganske dårlig. Da er det bedre å bruke en tradisjonell bok.

Når man skal designe et program for læring er det viktig at man utnytter datamaskinens egenskaper som interaktivitet, evnen til å reagere på input, muligheten til å vise animasjoner, utnyttelse av multimedia, evnen til å raskt finne data i store datamengder, evnen til å sette sammen data på mange forskjellige måter osv…



Målsetting

Bygg på din erfaring. Alle har erfaring fra en læresituasjon, enten som lærer eller som elev. De programmene som vi skal designe/prototype har to målgrupper - elever og lærere. Når vi skal prøve å identifisere emner som er egnet for et dataprogram, bør vi tenke tilbake på hvilke emner vi erfarte problemer med i forbindelse med egen læring/undervisning. Hensikten med å designe et program er jo nettopp å avhjelpe virkelige problemer hos elevene. Vi bør analysere hva som var vanskelig og hvorfor det var vanskelig.

Identifiser undervisnings/læringsproblem og prøv å finne ut hva elevens problem er. Har han konsentrasjonsvansker? Lider hun av ADHD? Motorikk? Er det problematisk å se sammenhenger? Krever emnet et høyt abstraksjonsnivå? Kjedelig? Feil «angrepsmåte» (læreren forteller, men eleven vil heller eksperimentere selv) etc...

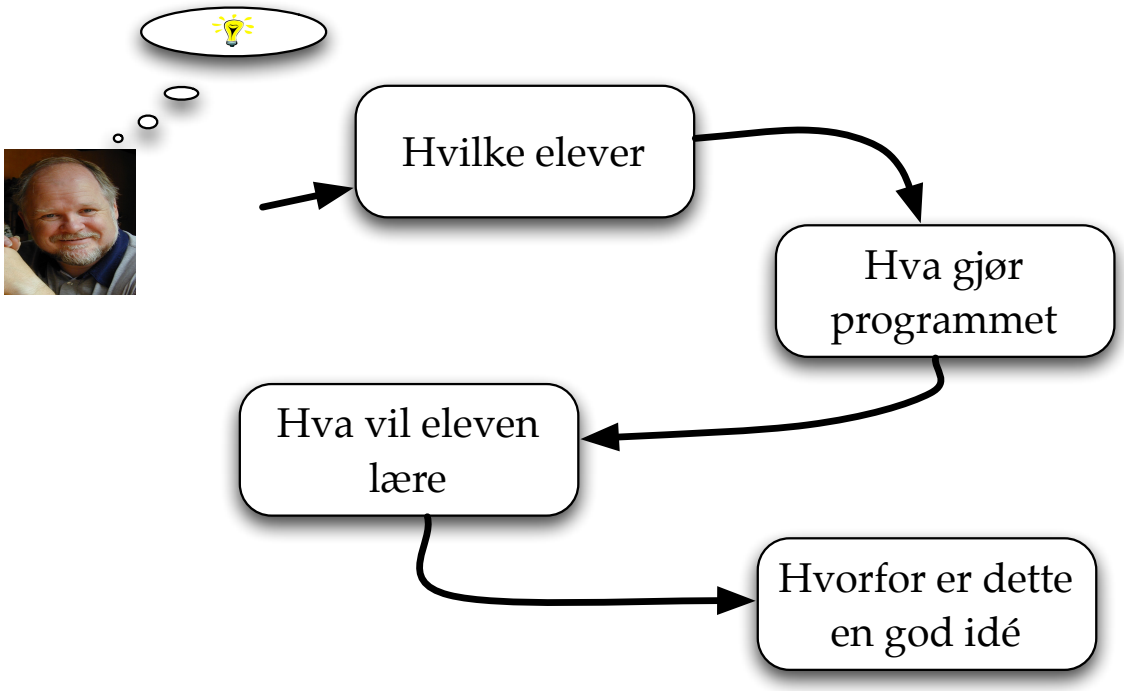
Og ikke minst - hvilke behov har eleven? Har eleven behov for mer tid? Trenger hun en annen fremstillingsmåte - kanskje hun er mer visuell, mens temaet vanligvis forklares mer logisk/skriftlig?

Hvilke sentrale begreper finnes? Hvis vi f.eks. ønsker at eleven skal forstå den historiske utviklingen i Europa i mellomkrigstiden er det viktig at hun vet hva Versailles-traktaten gikk ut på. Konflikten mellom Frankrike og Tyskland om de omdiskuterte områdene i Elsass/Lothringen (Alsace/Lorraine) blir en viktig byggestein. Og det er klart at det er en fordel om eleven vet litt om hva inflasjon er og hva den skyldes. Osv…

Det er svært mange ting som spiller inn i en læresituasjon. Noen av disse kan være:

- For lite utstyr
- For dårlig tid
- Dårlig motivasjon
- Stramt budsjett
- Farefullt
- Dårlig tilpasset lærebok

Dette er bare eksempler, og man kan nok komme på mange andre. Etter at ideen er formet er det mange vurderinger man må gjøre. Man må konkretisere idéen og identifisere hva eleven skal lære av å bruke programmet. Det er nødvendig at dette blir mest mulig konkret fordi man i det videre designarbeidet stadig vil bli stilt ovenfor alternativer der man må velge det som best oppfyller den målsettingen som er gitt for programmet. Jo mer konkret målformuleringen da er gjort, jo enklere vil det bli å ta disse avgjørelsene. Videre må man identifisere og beskrive målgruppen så konkret og eksakt som mulig. Forskjellige målgrupper vil kreve forskjellig utforming av programmet. Dette vil gi føringer på valg av metafor, presentasjonsform, dialogform etc… Alle disse vurderingene fører så til en målsetting.



Så må vi vurdere hvorfor vi skal bruke en datamaskin til dette? Hvilke fordeler har det i forhold til tradisjonell pedagogikk? Hvilke fordeler finnes f.eks. i forhold til andre media? Kan denne idéen realiseres enklere gjennom bruk av andre, mer tradisjonelle undervisningsmetoder?FInnes det alternative måter dette kan realiseres på uten å ta ibruk datateknologi? Å utvikle programvare er en ressurskrevende prosess. Hvis vi ikke kan forsvare idéen i forhold til overnevnte problemstillinger, skal man være forsiktig med å sette igang utviklingsprosessen.

Pedagogiske målsettinger med et program

Før man begynner arbeidet med å lage et program er det viktig å ha klart for seg hvorfor man utformer programmet slik man gjør. Et godt program kjennetegnes av at det er gjort bevisste designvalg underveis. For alle programmene vi utvikler i dette faget kan man angi en generell intensjon:

> ***Programmet skal, hvis det brukes på den angitte måten, bidra til at brukeren/brukerne lærer noe."***

På samme måte som for helsebringende produkter må man avklare:

- tiltenkt effekt
- bruk (riktig dosering)
- mulige bieffekter

Det er fristende å legge til at dette faller dødt til jorden hvis man har stilt feil diagnose!

Tiltenkt effekt

Den tiltenkte effekten er som nevnt at brukeren/brukerne lærer noe. Hva vil det så si å lære noe? Er det nok at man *husker*, eller skal man kreve at man skal kunne *anvende det lærte i en ny situasjon*? En definisjon kan være at eleven blir i stand til å utføre en handling han eller hun ikke var i stand til å utføre tidligere. Det å være i stand til å utføre en handling i en gitt situasjon kan vi kalle en **kompetanse.**

Ekesmpler på kompetanser:

- Skrive ned alle tyske preposisjoner som styrer dativ
- Lage tyske setninger muntlig der preposisjonene styrer dativ riktig
- Lage tyske setninger skriftlig der preposisjonene styrer dativ riktig
- Visuelt skille en bil fra en traktor
- Styre en bil
- Kjøre en bil i trafikken
- Kjøre en bil gjennom gatene i London en formiddag

Når man skal være konkret er det viktig å unngå setninger med ord som "forstå", "kunne", "vite" osv. som ikke angir hvilke *handlnger* eleven er i stand til å utføre.

Eksempler på **dårlig formulerte målsetninger:**

- Eleven skal kunne gangetabellen
- Eleven skal forstå bilens virkemåte
- Eleven skal kjenne de norske fylkene
- Eleven skal ha lært om sitronsyresyklusen
- Eleven skal ha kunnskap om 30-årskrigen
- eleven skal kunne den norske kongerekken

Bruk av programmet

Når man har angitt de pedagogiske målsetningene med programmet er det viktig å angi hvordan programmet er tenkt brukt. gode programmer lar seg bruke i mange forskjellige læresituasjoner, men det er likevel viktig både for lærer og elev å vite hvordan programmets forfattere har tenkt det brukt. Det er også ønskelig at det anslås hvor lang tid elevene stort sett bør bruke programmet.

Eksempler på anvendelser:

- En elev foran hver maskin uten verken introduksjon eller veiledning fra lærer. Intet skriftlg materiale. Dette er "Robinson Crusoe"-varianten: "Hvordan overleve alene på en øde øy".
- En elev foran hver maskin, men læreren har gitt en introduksjon.
- To elever foran hver maskin med
  - introduksjon
  - tilgjengelig video
  - lærer som går rundt
  - god lærebok
  - tilhørende øvingshefte
- 2, 3 eller 4 elever i gruppe foran en maskin med
  - foregående tavleintroduksjon
  - matchende lærebok
  - tilpasset øvingshefte med oppgaver av forskjellig vanskegrad
  - en god video til programmet
  - en engasjert lærer som er tilgjengelig
  - kun 15 elever pr. klasse
  - ressurser til ekskursjoner for å fylle ut med konkrete eksempler
  - tilgang på et godt bibliotek med bøker og relevante videoer
  - en godt utstyrt lab
  - et godt klassemiljø
  - behagelige lokaler
- Programmet er tenkt brukt av læreren på overhead-skjerm som et supplement

Mulige bieffekter

Med bieffekter menes både positive og negative bieffekter.

Eksempler på bieffekter:

- Positive
  - Elever lærer seg å bli mer digitalt kompetente (Hovedeffekten kunne være å bruke IT til å finne informasjon på nettet)
- Negative
  - Elevene kan få et uheldig instrumentelt forhold til dyr (Hovedeffekten kunne være å dissekere frosker)
  - Elevene får ikke trent på håndskrift (Hovedeffekten kunne være å lære å bruke tekstbehandling)
  - Elevene får ikke lagt hånd på konkrete objekter (planter, reagensrør, frosker, terninger…) fordi tiden brukes til å simulere disse på en datamaskin.

Fra pedagogiske målsetninger til pedagogisk program

Når de pedagogiske målsettingene med programmet er avklart, er tiden kommet til å finne hvilke virkemidler som skal tas i bruk for å oppnå ønsket effekt. Generelt kan man si at:

Den beste læresituasjonen for å oppnå en kompetanse er en situasjon som er mest mulig lik den livssituasjonen der kompetansen skal anvendes.

For barn fungerer lek ofte som en slik læresituasjon.

Virkemidler

Datamaskinen som medium tilbyr en del virkemidler som ikke finnes i trykt materiale. Det følgende er et forsøk på å liste opp en del virkemidler vi har tilgjengelig når vi benytter et dataprogram:

Datamaskinen som medium (ren gjengivelse):

- Tekst på skjermen. Lese/skrive
- Statiske bilder i sort/hvitt eller farge. Se/tegne/"scanne"
- Lyd. Innspille/avspille
- Tegnefilm. Oppleve/lage
- Video. Oppleve/spille inn.
- Styring av "dingser" utenfor datamaskinen
- Bruk av datanett mot eksterne data/andre personer
- Utskrift

Datamaskin som skaper av en interaktiv illusjon:

- Datamaskinen som arkiv. Passive data.
  - Navigering i, og ordning av store datamengder
- Datamaskinen som laboratorium og arbeidsbenk. Aktive data.
  - Simulering av matematiske/logiske modeller. Tenkt verden.
  - Bygging av noe v.h.a. byggeklosser eller formalismer.
- Datamaskinen som orakel, samtalepartner og lærer
  - Datamaskinen tester "riktig svar"/ "tygger" på noe du har skrevet inn eller laget. Herunder gjenkjenning av bilder og lyd.
  - Såkalte kunstig inteligenssystemer. Styrt tekstlig dialog.
- Veileder gjennom jungelen
  - Overbygg for å samordne alle virkemidlene.

Listen over er kun en enkel oversikt. Den prøver ikke å være komplett eller uttømmende, men er ment som et utgangspunkt for videre diskusjon.

Valg av virkemidler

Hva er så koblingen mellom pedagogiske mål og valg av effektive virkemidler? Dessverre har det vist seg at denne koblingen langt fra er ukomplisert. Det finnes ingen enkel oppskrift som angir det beste læreprogrammet ut fra en formulert målsetning. De fleste valg som tas er basert på "synsing" og generaliseringer ut fra vagt forståtte erfaringer. Det er også så store forsjeller mellom elevene at en rigid kokeoppskrift ville ha vært en absurditet.

Det følgende er en subjektiv liste av "bør" og "bør ikke", som oppsummerer noen erfringer og tanker:

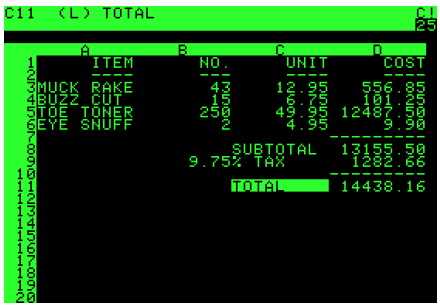
- Gjør programmet tiltalende og lett å bruke
  - Legg flid i å utforme layout som om det var en lærebok
- Ikke abstraher unødvendig
  - Vær konkret. Hva er elevenes begrepsverden?
- Bruk mange eksempler, også fra andre fag
  - Et materiale kan alltid sees fra mange flere sider enn man tror.
- Bruk lyd og bilde. Opplevelser gir nyttige knagger.
  - Et bilde sier mer enn 1000 ord. Levende bilder gir viktige opplevelser.
- Aktiviser eleven
  - Det er viktig å skape interesse og å motivere til egenaktivitet
- Gi elevene utfordringer
  - Påtving ikke elevene noe vanskelig, men la det finnes noen nøtter
- Ikke fortell elevene hvor dumme de er
  - Det er straff nok å ikke få noe til

- Unngå antropomorfismer
  - Datamaskinen er ingen person
- Sett kunnskap i en historisk/tverrfaglig sammenheng
  - Nei til objektiv "informasjon". Hvem mener hva?
- Gjengi en dialog/diskusjon om emnet
  - Få fram uenigheter. Det stimulerer tenkeboksene
- Lag skriftloig materiale også på papir-medium
  - Tekst gjør seg best på papir
- Legg inn selv-test muligheter.
  - Elevene ønsker å vite om de har lært noe/opparbeidet seg en kompetanse
- Kurskorrigering med jevne mellomrom under designprosessen
  - Det er svært lett å miste målsettingen av syne

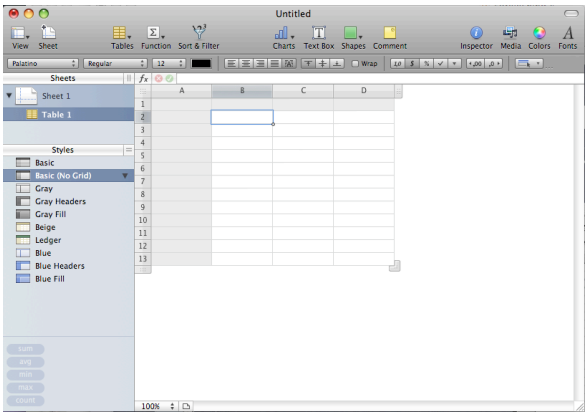
Når du har laget et design, prøv å sette deg i elevens sted. Tøm hjernen for alt du vet om emnet, slipp haken ned på brystet og stirr tomt fram for deg i total uforstand. Vær eleven. Hva er elevens arbeidssituasjon? Hva vet han/hun om faget fra før? Hvordan oppleves det for eleven å bruke det designet dere har laget? Hvordan ville dere ha reagert på et slikt program da dere var på elevens alder? Prøv å huske tilbake. "Bruk" programmet og opplev. Prøv å sette ord på det.

Metafor

Når man skal formgi et program er det ofte (alltid?) mulig å plassere brukeren inn i en verden - et sett med kjente omgivelser brukeren kan benytte for å orientere seg i, lære seg å bruke, programmet. En metafor kan beskrives som "den verden som er valgt som omgivelser for programmet", og benyttes vanligvis for å lette kommunikasjonen mellom programmet og brukeren. I tilleg kan man benytte metaforer for å øke motivasjonen hos brukerne. Det er mange program som benytter en metafor uten at vi tenker så mye over det. Et av de flrste var regnearket VisiCalc som ble utviklet av Daniel Bricklin og Robert Frankston i 1978. Metaforen er her en regnskapsbok der arkene er inndelt i celler som kan inneholde tall, tekst og formler. Stort sett alle program av denne typen som er utviklet siden har vært basert på denne metaforen. De mest brukte idag (MS Excel og Numbers) bruker fremdeles denne metaforen, selv om Numbers er i ferd med å bryte ut fra denne og benytte et lerret der man kan legge inn tabeller og andre objekter. Disse tabellene følger imidlertid denne, etter hvert, ganske gamle metaforen.



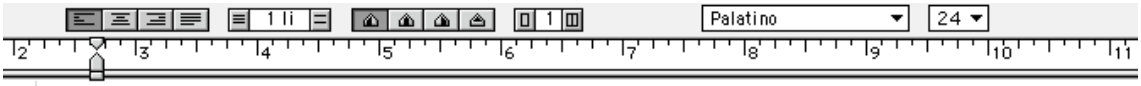
VisiCalc



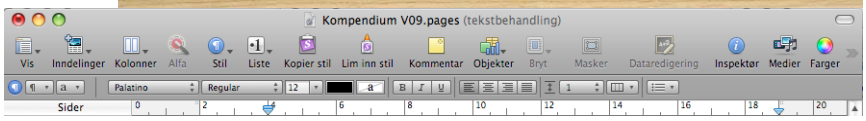
Numbers fra Apples iWork-suite

En annen metafor som etterhvert er blitt svært vanlig er "desktopmetaforen" som ble introdusert ved Xerox PARC på 1970-tallet og videreutviklet av Apple. Alle operativsystem idag benytter seg av denne metaforen når de presenterer filstrukturen for brukerne. Imidlertid fungerer ikke denne metaforen optimalt lenger. Den ble utviklet mens disker var små, og man forventet at brukerne selv skulle holde oversikten over hvor filene befant seg. Dette fungerte godt når man hadde noen hundre filer, men dagens maskiner har en helt annen diskkapasitet enn maskinene hadde tilbake på 80-tallet og denne metaforen er i ferd med å "gå ut på dato". Imidlertid har man foreløpig ikke kommet fram til noe som fungerer bedre.

Man finner også andre eksempler på mindre og mer begrensede metaforer i en del programvare. F.eks. utformer man gjerne noe som skal trykkes på slik at det ligner på en fysisk knapp. Dermed vet brukeren når han ser dette objektet at dette er noe han kan trykke på. Da de første GUI (Graphical User Interface) - baserte tekstbehandlerne dukket opp, benyttet de en linjal øverst i skjermbildet for å gi brukerne en enkel måte å stille inn tabulatorer og tekstjustering på. Dette fungerte godt på den tiden, da de som skulle bruke slik programvare var



vant med å skrive på skrivemaskin, og det var slik dette ble gjort på en skrivemaskin (skrivemaskinene hadde en "linjal" øverst der man stilte inn tabulatorer). Idag er ikke dette kjent for brukerne, og man ser at "linjalene" i moderne tekstbehandlere har endret seg i forhold til de tidlige inkarnasjonene.



Så hva som er en god metafor vil endre seg over tid, avhengig av hva som er brukernes forkunnskaper og forutsetninger.

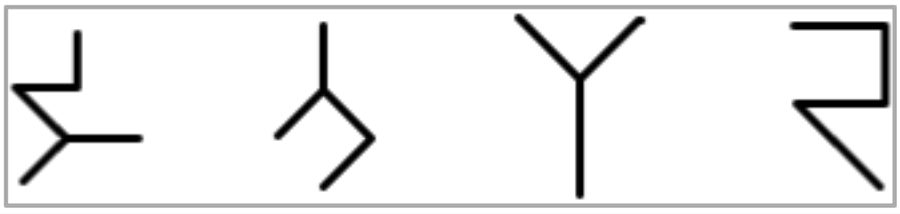
I pedagogisk programvare kan man tenke seg forskjellige metaforer. I et geografiprogram kan man f.eks. være en turist som reiser rundt i et land for å bli bedre kjent med det. Denne metaforen kan f.eks. også benyttes i et språkprogram. Matematikkprogram bruker ofte forskjellige spillmetaforer - Yatzy, forskjellige brettspill eller lignende. Hvilken metafor vi velger vil ha store konsekvenser for hvordan programmet vil ta seg ut på skjermen, og for hvilke aktiviteter eleven vil utføre under programkjøringen.

Eksempel

Vi skal lage et program som skal trene elevene i logikk og mønstergjenkjenning (noe som mye læring dreier seg om). Idéen er å presentere flere mønstre og trene elevene i å se hvilke mønstre det er enklest å omforme til et annet. Man skal lage figurer og endre disse på en mest mulig effektiv måte. Poenget er å se sammenhenger mellom forskjellige strukturer og lage seg en strategi for hvordan et mønster mest mullig effektivt kan omformaes til et annet. Et slikt problem kan plasseres inn i flere metaforer. I det følgende presenteres 2 (av mange) mulige løsninger:

Pinnespillmetaforen

Vi kan implementere dette som et pinnespill. Dette blir som å lage et spill der man bygger figurer av fyrstikker.



Man presenterer

oppgavene som kort der det er tegnet pinnefigurer. Alle pinner er forbundet med minst en annen pinne, og alle pinner danner en vinkel på 45° med hverandre. Antall pinner i et mønster kan være fra 1 til 6. Et trekk defineres som å ta vekk eller å legge til en pinne fra mønsteret. Å rotere en pinne rundt et festepunkt regnes ikke som et trekk.

Et spill kan beskrives slik:

4 kort (med bilde av et mønster på hvert av dem) snus. Deretter velges et av disse og mønsteret konstrueres. Når man har utgangsmønsteret, velger man et av de resterende 3 synlige mønstrene og omformer det man nettopp har konstruert til det nye. Når man har konstruert ferdig, snus et nytt kort slik at det er 3 valgbare mønstre synlig. Slik fortsetter man til det ikke er fler mønstre igjen.

I denne metaforen får vi et ganske oversiktlig skjermbilde som er lett å stilisere. Det blir lett å forholde seg til, og det vil ligne på andre verktøy som eleven er vant til å bruke, men det vil kanskje oppleves av noen elever som litt kjedelig.

Labyrintmetafor

En alternativ metafor kan være et «hulespill» der man har som oppdrag å finne fram i en labyrint. Oppdraget blir at man skal redde prinsessen som befinner seg i et låst rom et eller annet sted. Man kommer inn i labyrinten der man ser tre dører. For å komme videre må man lage en nøkkel som man kan bruke for å åpne en av de tre dørene. Når man kommer gjennom denne døren kommer man inn i et nytt rom med 3 dører etc…

Som alle kan skjønne må programmet se veldig forskjellig ut i disse to metaforene. I pinnespillmetaforen jobber vi på et lerret der det er svært få forstyrrende elementer. Vi vil ta tak i de forskjellige elementene med den vanlige markøren og jobbe direkte med denne. I labyrintmetaforen vil det være ganske mange kulisser, og for at dette skal føles naturlig, vil man velge i en verktøykasse avhengig av hva man skal gjøre. F.eks. vil man kanskje ha en tang for å rotere et nøkkelelement, en skjærebrenner for å løsne et element og et sveiseapparat for å feste det. I denne metaforen vil det være mer naturlig å ha bevisste verktøyvalg, verktøy som må illustreres på skjermen på en eler annen måte.

Pinnespillmetaforen vil gi et renere design med færre forstyrrende elementer, mens labyrintmetaforen vil føles mer som et tradisjonelt spill og vil kanskje fungere motiverende for en gruppe elever.

Hvilken av disse metaforene som vil være den beste vil være avhengig av brukerne. Det er ikke gitt at den ene nødvendigvis er bedre enn den andre. Hvilken som vil fungere best vet vi ikke før vi har testet dem på brukerne. Derfor er det viktig å kunne bygge prototyper og teste ideene ut på brukerne.

Tid, sted og rolle

Hvordan skal vi så komme fram til disse forskjellige metaforene? Idéer til gode metaforer vokser ikke på trær. Det er her disse tre begrepene kommer inn. Avhengig av den metaforen som er valgt, vil brukeren få tildelt en rolle han skal «spille» under kjøring av programmet. Dette vil så også legge føringer for hvordan tid og sted blir behandlet i programmet. Eller vi kan snu det på hodet, og ta en diskusjon om bruk av tid, sted og rolle i programmet, og se om vi kan få noen gode oideer til en metafor ut fra en slik diskusjon.

Utgangspunktet for denne diskusjonen må alltid basere seg på den tenkte målgruppen. En metafor skal være noe som målgruppen kan benytte for å lette kommunikasjon med programmet, eller for å bedre motivasjonen til medlemmer i målgruppen. Men da må vi alltid ha målgruppen i bakhodet. Det er ingen vits i å bruke en metafor som passer for elever i 2. klasse v.g. skole hvis man er ute etter å lage et program for en 5. klassing.

Det er også viktig å tenke på at det aldri er bare en metafor som kan passe. Når vi designer, er vi på utkikk etter den beste løsningen. I dette ligger bl.a. at vi må prøve å identifisere så mange metaforer som mulig for det gitte programmet, slik at vi har en mulighet til å velge ut den som vil passe best. Man finner sjelden den beste løsningen i første forsøk, og det er liten grunn til å tro at den første metaforen som en kommer på vil være den beste. Ofte har vi forutinntatte idéer om hva som vil fungere best, og det er svært vanskelig å gi slipp på slike ideer. Man har lett for å tenke at brukeren er som meg, og det jeg liker vil også brukeren like. Slik er det dessverre ikke.

For å hjelpe oss i arbeidet med metaforvalg kan vi undersøke hvordan vi kan manipulere de tre elementene tid, sted og rolle i programmet. En av datamaskinens egenskaper er nettopp at vi bygger opp en simulert virkelighet på skjermen der man kan endre f.eks. tiden. Noen emner der man kan ha behov for å endre på tidsbegrepet kan være prosesser som tar svært lang tid i virkeligheten, f.eks. forsøk innen arvelære der det vil ta lang tid å få fram flere generasjoner slik at man kan se resultatene. I et slik tilfelle kan man manipulere tiden slik at den går fortere slik at man i løpet av nien minutter kan se hva som ville ha skjedd 7 generasjoner inn i fremtiden. Et annet eksempel kan være reiser, der man i løpet av noen få minutter kan besøke land som det vil ta mange dager og uker å besøke i virkeligheten. Andre måter å manipulere tiden på i et dataprogram kan være å få den til å gå langsommere (for å se hva som egentlig skjer i noe som skjer veldig fort - f.eks. for å vise hvordan lukkeren i et SLR-kamera med LiveView fungere), eller også stoppe den. Tiden kan gå stegvis, vi kan se på fortid og framtid, eller vi kan se på den dublert (f.eks. gjøre et forsøk med 2 forskjellige instillinger på en parameter og så se på forløpet av forsøket i hvert sitt vindu på skjermen samtidig).

Men diskusjonen om hvordan vi ønsker å behandle **tiden** må også sees i sammenheng med de to andre elementene: **sted** og **rolle**. En diskusjon av hvordan vi vil behandle tiden i programmet vil kunne ha føringer for de to

andre parametrene. Det stedet «handlingen» i programmet foregår må diskuteres. Vi må også diskutere elevens synsfelt inn mot dette stedet. Ser eleven noe ovenfra (i fugleperspektiv) eller nedenfra? Det finnes minst like mange alternative synspunkter enn det finnes måter å behandle tiden på. Man kan tenke seg at programmet viser en virkelighet i et annet land, eller at det foregår ombord på en båt eller i et fly. Men man kan også tenke seg mer uvirkelige synspunkter som f.eks. at man ser et snitt gjennom jordkloden, eller at man ser ned på jorden med markerte grenser (som i GoogleEarth). Man kan se noe mikroskopisk som man ikke kan se ellers (et virus/en bakterie) og bevege seg på steder der man ikke kan normalt (inne i en vulkan, eller inne i blodomløpet til en person). En av styrkene til datamaskinen i en pedagogisk sammenheng er jo nettopp at man kan fjerne seg fra virkeligheten og gi eleven innblikk i et fenomen på en måte man ikke kan ellers.

Dette kommer også an på den rollen vi lar eleven spille i programmet. Hvilke hjelpemidler vil være tilgjengelig for den aktuelle rollen? Vil det være naturlig for en person i denne rollen å se det som vises? Ekesmpler på roller kan være en annen person, den personen som skal studeres, en person som inngår i det emnet som skal studeres, et dyr, et insekt, en ting etc… Man kan f.eks. tenke seg et program der man skal lære om norsk geografi plassert inn i et program der man benytter postverket som metafor. Rollen kan da vøre en av postverkets «detektiver» som skal finne fram til poststeder basert på kanskje mangelfullt adresserte brev. Naturlige hjelpemidler i en slik setting ville være et kart over Norge, en bok med oversikt over postnummer, en grafisk database med detaljkart over norske kommuner etc… Denne metaforen er svært konkret, noe som er positivt, men man må søke å stilisere de elementene som er med i metaforen slik at ikke skjermbildene blir overlesset med unødvendig informasjon og kulisser som bare har som funksjon å understøtte metaforen og som ikke har noen funsjon i seg selv.

Man kan tenke seg dette lagd som et spill, men det er avhengig av brukergruppen. Det er ikke alle brukere som blir motivert av et spill. Dessuten er elevene idag så vant med avanserte dataspill at noe slikt fort vil kunne fortone seg som kjedelig.

Alle disse elementene må smelte sammen til en troverdig og helhetlig **metafor** som virker naturlig og som hjelper eleven i bruken av programmet.

Eksempler:

Idéen kan være å lage et program omkring kystfiske. Å lage et program som belyser alle sider ved dette vil være vanskelig. Man kan ønske å dra inn økologiske eller økonomiske spørsmål, politikk, bosetning, biologi etc… Man må allerede her gjøre en avgrensning. Avhengig av hvilken vinkling man velger vil forskjellige roller passe inn. Roller som kan være aktuelle er «Firskeskipper», «Forsker», «Økolog», «Lokalpolitiker», «Bellona-aktivist» mm… Måten man vil behandle tiden på vil endre seg ut fra den vinklingen man velger. Man kan se på kystfiske i fortid, eller se på utviklingen og simulere flere forskjellige mulige

utviklinger for kystfiske avhengig av valg man gjør. Man kan se på en parallell tid der man lar et lokalsamfunn utvikle seg med forskjellige forutsetninger. Man kan komprimere tiden og se på mange års utviklng i løpet av noen få minutter etc… Alternativene er mange, og det er viktig at vi vurderer alle sammen slik at man har en god forutsetning for å ta et valg.

Hvor skjer dette? Hvilket sted/synspunkt velges for programmet? Det vil igjen være avhengig av de andre valgene vi har gjort.

Legg merke til at disse begrepene krever at man er **konkret** når man beskriver designet. Hele designmetoden vektlegger dette med å være konkret, å konkretisere de idéene og begrepene vi vil at elevene skal lære av å bruke programmet. Dette er en av bærebjelkene i hele designmetoden.

Aktivitetstabell

Etter at man har diskutert metaforer og valgt ut den man ønsker å benytte, er tiden kommet til å konkret beskrive hva de forskjellige aktørene skal gjøre. Det er viktig å aktivisere elevene mens han/hun bruker programmet. Dette gjelder både fysisk aktivitet (sørge for at eleven aktivt gjør noe i skjermbildet) og mental aktivitet (eleven skal tenk gjennom det som skjer og ta aktive valg, ikke bare slavisk følge noe som beskrives på skjermen). Man setter da opp en **aktivitetstabell** (eller kanskje **ansvarstabell**) som viser hvem som har ansvaret for at en aktivitet blir utført. Dette vil påvirkes av hvilken målgruppe som er valgt, hvilken målformulering vi har vagt og hvilken metafor vi har planer om å bruke. Disse aktivitetene kan være synlige (menyvalg eller knapper man trykker på) eller usynlige (skrive i et felt eller flytte noe rundt på skjermen).

Beslutninger man må ta her kan være:

- hvem bestemmer noe
- hvem velger noe
- hvem forandrer noe
- hvem skal bestemme hva som skal noteres/lagres
- hvem lager noe
- hvem kontrollerer det som skjer

Disse spørsmålene kan sammenfattes i to sentrale spørsmål:

- hvem blir aktivisert?
- Hvem får ansvar?

I denne fasen tar vi ikke stilling til hvordan disse aktivitetene skal utføres, bare at dette er aktiviteter som skal inngå!

Mange design feiler i dette - eleven blir en passiv tilskuer. Det første målet vi må ha når vi skal lage et program for læring, er at eleven blir aktivisert og deltar aktivt i læringen, både fysisk og mentalt. Da kan elevens interesse bli vakt og dermed kommer også motivasjonen. Her kan det passe å komme med et "ordspråk": " Det jeg har hørt, tror jeg; det jeg har sett, tror jeg; men det jeg har gjort, **kan** jeg." Et av problemene med den tradisjonelle undervisningen er at elevene lett blir passive og ikke deltar aktivt i timene. Samtidig er en av de egenskapene vi mener gjør datamaskinen til et effektivt læringsinstrument nettopp at den er interaktiv slik at elevene kan føre en "dialog" med den. Eleven skal altså gjennom en dialog med programmet arbeide seg fram til erkjennelse. Dette forutsetter aktivitet fra elevens side. Det er designerens ansvar å sørge for at eleven blir aktivisert.

For å sikre oss at dette skjer må vi beskrive de forskjellige aktivitetene som eleven skal ha ansvar for i en aktivietstabell. Man kan tro at man beskriver et

program som forutsetter en aktiv elev, men ofte er ikke dette tilfelle.. Dette ser man lettere hvis man konkret skrivewr ned på papiret de aktivitetene man tenker seg aktørene har. I en aktivitetstabell skal vi i tillegg beskrive hva eleven skal gjøre, som *tegne figurer, plassere tall, trekke verb på rett plass i setningen, slå opp i kartotek etc…*

Et par eksempler:

Forslag til aktivitetstabell for et program for barnehage der barna skal lage et tegning av seg selv.

Elev	Program
Skape et bilde av seg selv	Tilby et sett med ansiktstrekk
Velge ansiktstype, -farge, -trekk	Nekte å plassere noe på feil plass
Plassere ansiktstrekk	Vise en “film” av prosessen
Flytte ansiktstrekk	…
Fjerne ansiktstrekk	
Tegne ansiktstrekk fritt	
…	

Matematikkprogram om prosentregning:

Elev	Program
Oppgi max-verdi	Justere skala
Flytte %-slider	Beregne nye verdier
Be om forklaring	Vise vindu med forklaring
Få nye tall	Nekte å slippe tall på feil plass i formelen
Flytte tall inn på rett sted i formelen	Oppjustere formel
Gå til annen del av programmet	
Lukke deler av vindu	Lukke/vise fram nye deler
…	…

Erfaringsbilde

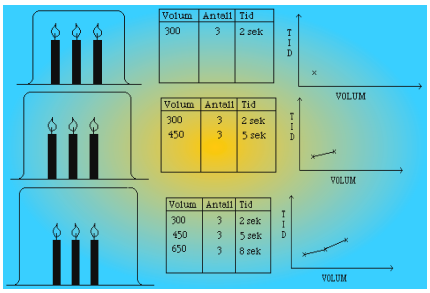
Mange mener at man lærer gjennom bruk av mentale modeller (jmfr. Donald Norman). Når vi opplever noe, prøver vi å forstå dette gjennom en modell vi bygger opp inne i hodet. Denne modellen tilpasses slik at de gitte faktaene passer. Vi kan se på denne mentale modellen som et "bilde" av det vi har opplevd. Modellen er ikke nødvendigvis korrekt, men hjernen prøver å konstruere denne slik at de fakta som oppleves passer inn. Vanligvis vil man ta en allerede eksisterende modell og prøve å tilpasse denne slik at de nye faktaene passer inn. Vi må sørge for at den mentale modellen våre elever bygger opp blir mest mulig korrekt. Derfor er det viktig at vi fokuserer på de elementene i programmet som eleven skal bruke som byggestener i denne modellen. Det er disse elementene vi kaller "erfaringsbilde".

Erfaringsbildet er de elementene som inneholder de sentrale, læringsbærende elementene i programmet. Det er de elementene som har til oppgave å

- gi elevene tilbakemelding på det de gjør
- integrere vanskelige konsepter
- vise akkumulerte effekter

Det er viktig å skille melom disse og de kulissene som programmet benytter seg av. Det er viktig å konsentrere seg om disse, og å finne hvilke elementer man ønsker å benytte. begrepet erfaringsbilde hjelper oss å være konkrete ved at vi må arbeide med å isolere, å finne, disse sentrale elementene.

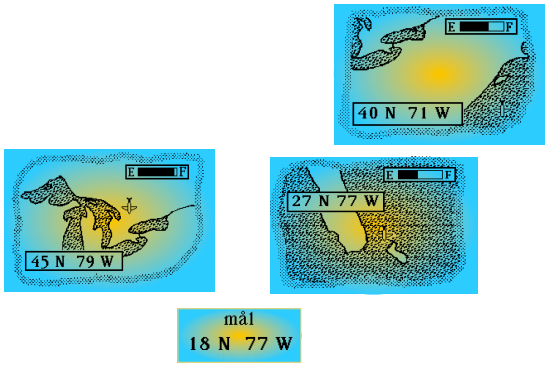
Eksempler



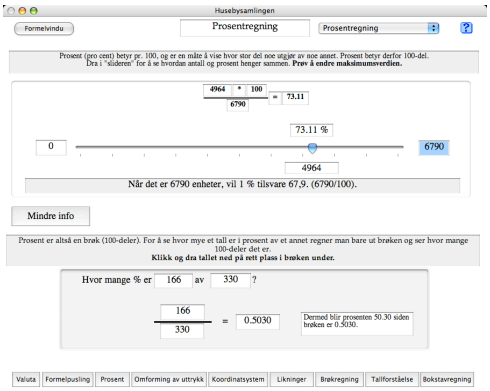
Dette er en fysikksimulering der elevene skal undersøke hvilke effekter som påvirker brenntiden i et lukket kammer. Metaforen som er valgt er et fysikklaboratorium der eleven gjør forsøk. Man kan endre volumet på luftklokken, antall lys og høyden på lysene (og kanskje en del andre parametre).

De sentrale elementene her bir å vise selve forsøket (#lys, høyden på lysene og størrelsen på luftklokken) samt tabellen som oppsummerer forsøkene, og grafen som viser tabellen i grafisk form. Hvordan resten av skjermbildet ser ut er sekundært. Om man tenner lysene ved å bruke en fyrstikk, eller bare trykker på en knapp spiller ingen rolle - det er ikke sentralt for læringsutbyttet.

Eksempel 2 - Et program for å forstå sammenhengen mellom lengde og breddegrad og plassering på et kart. Dette kan illustreres på mange forskjellige måter, men man har valgt en metafor der brukeren skal være en pilot som flyr hjelpesendinger til forskjellige steder på kloden. For å komme seg dit han skal får han oppgitt et målpunkt i lengde- og breddegrader og må navigere seg dit



ved å styre flyet. De viktige elementene som utgjør erfaringsbildet vil da være et kart med en indikator som viser hvor på kartet man befinner seg. Man må også hele tiden kunne se målpunktet og en indikasjon på hvilken posisjon man har i øyeblikket. Man kan vise ytelse (og gi en viss stressfaktor) ved å vise en bensinmåler som viser hvor mye bensin man har igjen (og kanskje om det er nok til å nå målet). Dette er de viktige elementene i designet - alt annet er sekundært for selve læringen. Hvordan selve flyvingen utformes, hvordan det ser ut i kartrommet og hvordan eleven får hjelpemeldinger er uvesentlig iden store sammenhengen. Det må selvfølgelig gjøres på en fornuftig måte, men det er ikke det som er viktig for selve læringen.



Eksempel 3 - et program for å lære om prosent. Her har man valgt en elektronisk arbeidsbok som metafor. Eleven jobber som han ville gjort i en bok, men man får tillegg som muligheter for å bruke animasjoner, og å få tilbakemelding om man gjør rett eller feil osv…

De viktigste læringsbærende elementene her er:

Formelen der tallene settes inn:

$$\frac{5021 * 100}{6790} = 73.95$$

Slideren som viser at prosent er et forholdstall:

En forklarende tekst som forklarer hva slideren viser:

Når det er 6790 enheter, vil 1 % tilsvare 67,9. (6790/100).

Et par ting…

Før en går videre i arbeidet kan det være lurt å stoppe opp og vurdere det som er gjort. Det er en del ting man bør passe på. Ofte er det sider ved et design man overser. Kanskje har man begynt å overlesse programmet med detaljer som egentlig hører hjemme et annet sted? Kanskje har man en del ting man gjerne skulle hatt med,men som man ikke helt har klart å plassere? Det kan da være fristende å lage en stor introduksjon, eller å fylle ut en del skjermbilder med masse tekst. Dette er ingen god ide, da det vil gjøre arbeidet mer uoversiktlig for elevene, og vil kunne virke forvirrende.

Noen ganger kan man bli fristet til å gjøre noen "farlige" forutsetninget. Man har kanskje noen elementer man ikke klarer helt å finne en plass til og bare legger dem inn likevel og si at man forutsetter at eleven vil få hjelp av læreren under programutføringen. Dette vil normalt ikke være tilfelle. Læreren vil ofte være opptatt med å hjelpe andre elever, og i andre tilfeller vil eleven bruke programmet alene uten at det er andre som kan hjelpe til stede. Et annet poeng er at hvis eleven ikke kommer videre i et program vil han bli sittende uten å gjøre noe og uten å si fra. Vi må derfor tilstrebe å lage et så klart og tiltrekkende design at elevene klarer å bruke det uten noen instruksjon, og at de også har lyst til å bruke programmet. Man må ikke designe program som krever store forklaringer for å kunne brukes.

Det er mange forhold som spiller inn i en læringssituasjon. Mange tror at det at undervisningen skjer ved hjelp av et dataprogram er motiverende i seg selv. Dette var kanskje et poeng tidligere, men idag er elevene så vant med moderne teknologi at det å bruke en datamaskin er noe dagligdags og ikke noe man blir motivert av. Da kan det heller virke demotiverende fordi det de bruker datamaskinen til privat ofte vil være langt mer underholdende og spennende enn det man bruker den til på skolen. Man må prøve å lete etter ting som kan virke motiverende for elevene. I alle fall må man passe på at man ike får inn elementer som virker direkte forstyrrende. Ting man bør tenle på er bl.a. at man sørger for at programmet oppfører seg konsistent. Med dette menes det f.eks. at brukergrensesnitet er konsistent, at programmet holder seg innenfor den valgte metafore,. at man ikke innfører fremmedelementer o.l. Man må også passe på at skjermen ikke blir overlesset med unødvendige kulisser. Dette har sammenheng med det som tidligere er sagt om stilisering av meytaforen.

Oppsummering

Målsetting

Kjenn brukeren, hans eller hennes forutsetninger og behov. Vær bevisst hvilket problem det er du er ute for å løse. Jo mer konkret du greier å være i dette, jo enklere jobb får du i det videre designarbeidet.

Metafor

Vurder flere alternativer for å finne den som passer best i forhold til brukeren og det man ønsker å oppnå. Det vil alltid finnes flere alternative metaforer, og det er viktig at man ikke ukritisk "hopper på" den man tenker på først.

Aktivitetstabell

Bruk dette verktøyet for å komme fram til hvilke aktiviteter brukeren skal utføre i programmet. Det er viktig at eleven blir aktivisert, men disse aktivitetene må være relevante i forhold til brukerens forventninger og forutsetninger. Alle aktiviteter som skal utføres skal ha en funksjon i forhold til den ønskede læringseffekten. Man skal ikke legge inn masse aktiviteter som ikke er relatert til det faglig innholdet bare for å aktivisere eleven. Da risikerer manbare at eleven blir forvirret eller kaster bort tid på aktiviteter som er irrelevante i forhold til læringsinnholdet.

Erfaringsbilde

Pass på at dere vet hvilke elementer i programmet som er viktige i forhold til det eleven skal lære.

Vi er nå ferdig med idéfasen, men vi mangler en oversikt over idéen. Vi har heller ikke kontroll med strukturen i designet og vi har enda ikke begynt å designe hvordan skjermbildene vi skal lage ser ut. **Vi har** (forhåpentligvis) masse **idéer om**

- hva som skal læres
- hvem eleven er
- hva eleven skal gjøre
- hva eleven skal oppleve
- hvilke elementer som skal være med

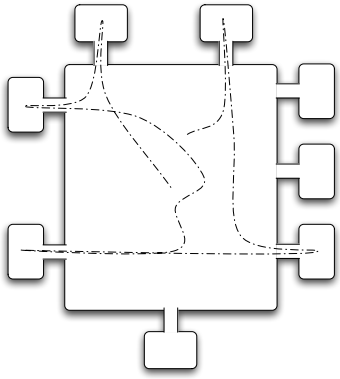
Men **vi vet ikke**

- hvordan ting henger sammen
- hvordan skjermbildene skal se ut
- hva som er avhengig av hva
- hva som skjer i alle situasjoner når en elev gjør noe i programmet.

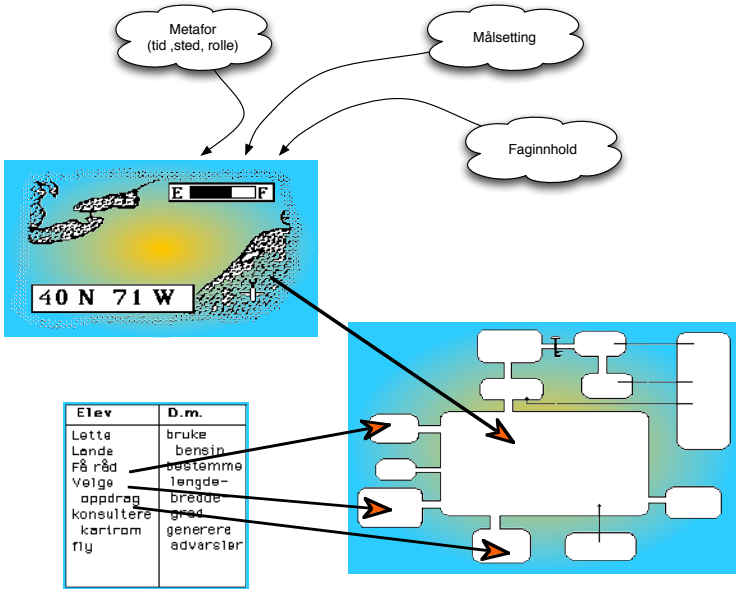
Vi trenger et verktøy som kan gi oss en oversikt over designet. Et som kan vise oss sammenhenger og som vi kan bruke for å få en oversikt over hva som påvirker hva og hva vi kan gjøre når. Dessuten må vi ha verktøy vi kan bruke for å få en følelse med hva som skjer og hva som endrer seg under programkjøringen.

Torg/Markedsplass

Vi trenger et verktøy som, på en side, kan gi oss en oversikt over designet vårt. I denne modellen har man valgt en markedsmetafor for dette. Et marked er ikke lineært - det er en stor åpen plass der man kan gå hvor man vil, i hvilken rekkefølge man vil, og så mange ganger man vil. Dette er en god metafor for hvordan vi ønsker at et pedagogisk program skal fungere. Eleven skal kunne følge sin egen vei i programmet og finne fram til de tingene han har bruk for. Hvis det er noe han ønsker å se litt mer på, skal han ha muligheten til det.

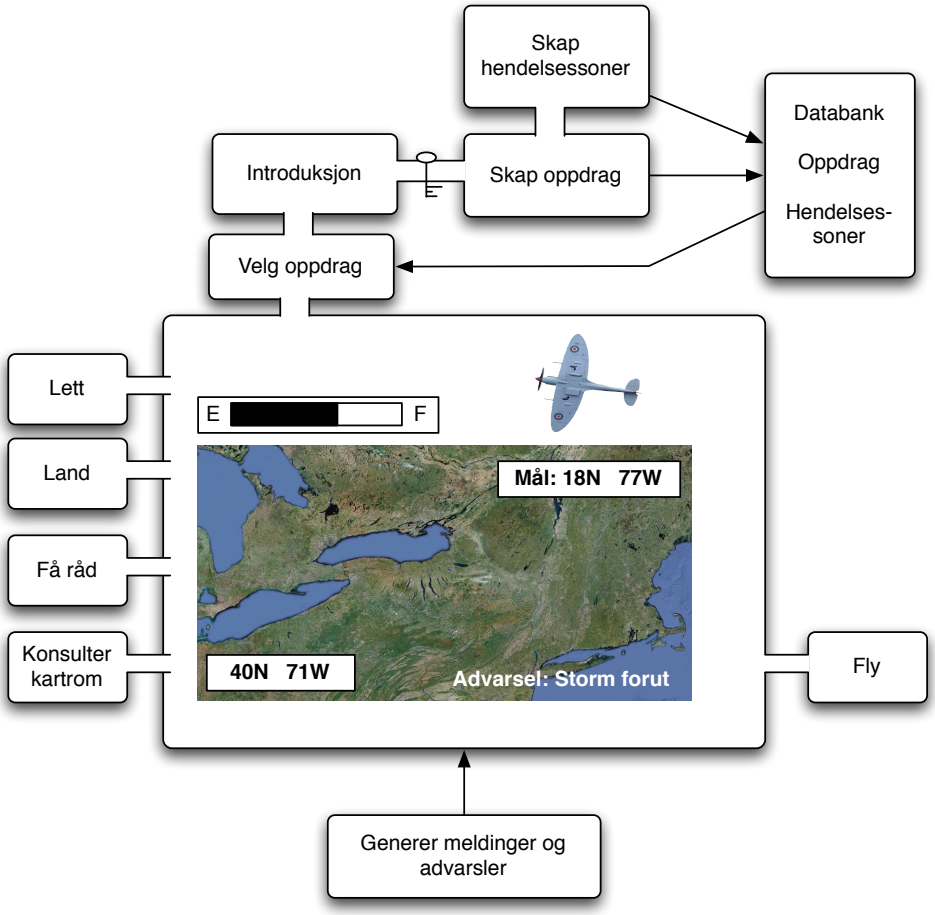


Et slikt program gir brukerne den frihet man trenger og ønsker for å gjøre ting på sin måte. Det gir større utfordringer og krever aktivitet fra eleven. Denne notasjonsformen er brukt for å tvinge designerne til å tenke mer fritt. Det er lett å ende opp med å tenke lineært. Det er enklest for designeren, men det er ikke den beste måten å designe på. Markedet skal være et verktøy for å organisere formgivergruppens tanker. Det er ikke et verktøy som skal gi en detaljert beskrivelse av programmet. Alle sider ved designet som er diskutert i gruppa skal komme sammen i markedet.



Erfaringsbildet påvirker modellen. Dette finner vi igjen på den sentrale markedsplassen. De elevaktivitetene som er beskrevet i aktivitetstabellen finner vi igjen som boder på torget.

Eksempel på et marked:



Dette er en tidlig versjon av markedet for et program for å lære om lengde- og breddegrader. Legg merke til at de elementene designerne har ment er viktig for læringen ligger inne på markedsplassen. Aktivitetene ligger som boder rundt markedet. Det er lagt en nøkkel på veien mellom "Introduksjon" og "Skap oppdrag" for å indikere at for å kunne legge inn nye oppdrag må man ha et passord. Det er ikke tatt stilling til hvordan noen av aktivitetene skal utføres. Legg merke til at alle aktivitetene som finnes i bodene er beskrevet med aktive verb - de betegner aktiviteter som elevene utfører, ikke noe maskinen gjør med brukeren som en passiv tilskuer. Markedet legger ingen binding på rekkefølgen man velger å gjøre ting i, og man står fritt i å velge hva man vil gjøre. Dette gjør at markedet ikke er en fullgod beskrivelse av logikken i et design, det er mer et ***mulighetenes marked.*** Man trenger altså et annet diagram for å spesifisere logikken og sekvensene designet inneholder. Til dette benyttes et tilstandsdiagram som vi skal komme tilbake til siden.

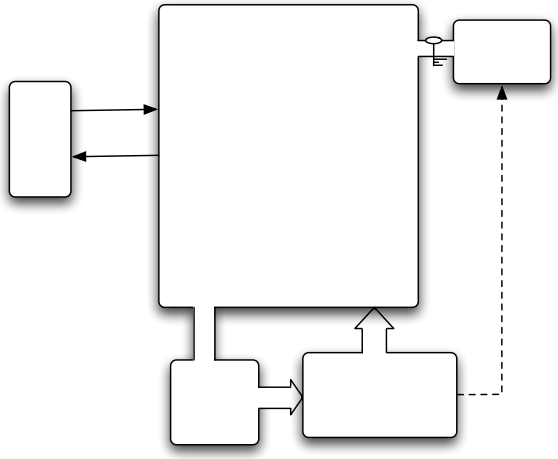
Et annet eksempel:

Prosentregning:

Dette er markedet for en del av programmet "Husebysamlingen" som er en samling program for bruk i matematikkundervisningen i 9. klasse på ungdomstrinnet.

Torgets syntaks

Selve markedsplassen markeres som et rektangel med avrundete hjørner. Mens et åpent valg er en boks med en forbindelse som er åpen i begge ender. Dette indikerer et valg som er tilgjengelig når som helst. Hvis man ønsker å gjøre dette valget tilgjengelig kun under spesielle brtingelser, markeres dette med en nøkkel over stien.



En sekvens markeres med piler mellom valgboksene.

Hhvis et valg har en innflytelse på et annet valg, markeres dette med en stiplet linje, mens dataflyt inn og ut av en boks indikeres med en heltrykket pil. Pilretningen indikerer hvilken vei datene flyter.

Hva torget er, og hva det ikke er

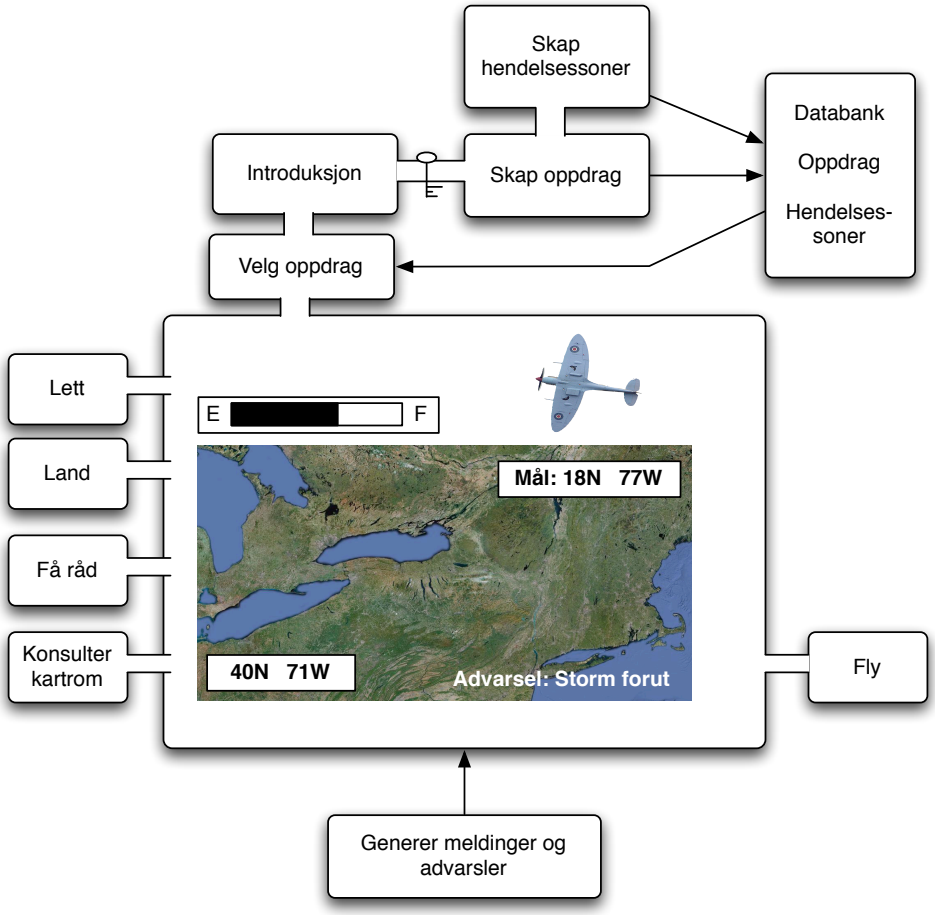
Torget er et verktøy for å få en oversikt over designet på et enkelt ark. Det kan brukes for å se sammenhenger mellom de aktivitetene vi har definert, og det hindrer en i å lage lineære løsninger som eleven ofte vil finne kjedelige.

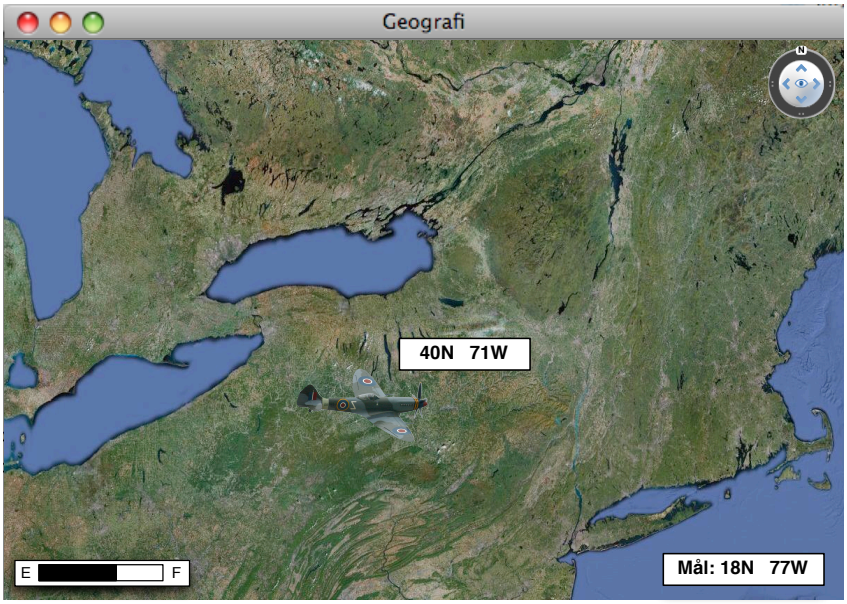
Torget er **ikke** en detaljert oversikt som vise formelle sammenhenger. Det er ikke et dataflytdiagram og slettes ikke en sekvensbeskrivelse.

Det er viktig at man behandler torget som det det er og ikke prøver å bruke det til noe det ikke er designet for!

Nøkkelskjerm

Når man har god nok oversikt over designet til at man har fått satt opp et markedsdiagram er det på tide å se på skjermbilder. Hva skal legges inn på hovedskjermbildet, hva skal plasseres inn i dialogbokser, hvordan skal kommandoene gis etc…



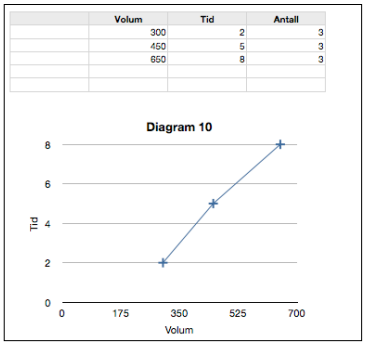


Det viktigste av skjermbildene i programmet kalles **nøkkelskjerm**. Det er på dette skjermbildet den viktigste læringen forsgår. Det er her vi finner de elementene som inngår i **erfaringsbildet**. Et program kan ha mange vinduer, men det vil alltid være et hovedskjermbilde. Andre vinduer vil gjerne designes som dialogbokser og vil spille en underordnet rolle. Som en konsekvens av dette er det dette skjermbildet eleven vil se oftest på skjermen - det er her eleven vil tilbringe mest tid. Grunnen til at dette begrepet er innført er for å få designerne til å fokusere på å designe et mest mulig effektivt arbeidsvindu og ikke bryte opp elevens arbeidsflyt ved å skifte skjermbilder og legge inn mengder av dialogbokser og paletter. Hvis man skifter skjermbilde for ofte kan det bli vanskelig for eleven å konsentrere seg. Man må da skifte mentalt fokus ofte, og dette krever både tid og oppmerksomhet og kan skape forvirriing. Man skal ha et vindu som kan fungere som et ankerpunkt man kan konsentrere seg om og som det er enkelt og raskt å komme tilbake til.

Vi begynner med å designe nøkkelskjermen først fordi det er her de vanskelige pedagogiske spørsmålene må løses. Hvis man ikke klarer å lage en god nøkkelskjerm har det ingen hensikt å lage andre vinduer/dialogbokser.

Nøkkelskjermen skal være en base elevene bruker for utforskning av det emnet programmet omfatter. Dat skal være en aktiv **gjøreplass** der elevene utfører aktivitetene vi har spesifisert i aktivitetstabellen. Den skal sørge for at elevene får tilbakemelding på det de gjør, og den skal sørge for integrering av de opplevelsene man har til større konsepter.

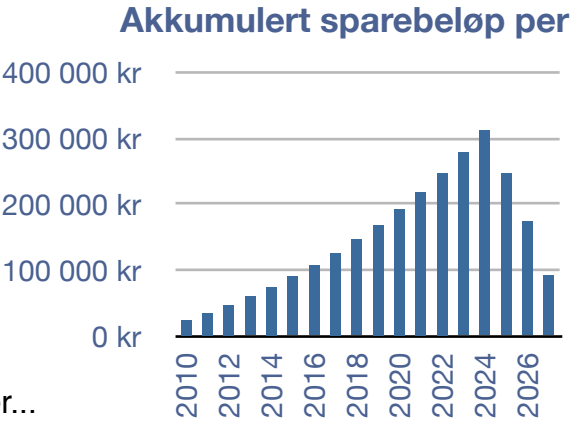
Det er mange elementer man skal integrere på en nøkkelskjerm. Den skal vise hva man har valgt, hva man ønsker å gjøre, hva man har gjort, og den skal vise relevant informasjon på en slik måte at elevene har mulighet for å skjønne at den er relevant. Den skal oppsummere hva man har gjort så langt, den skal måle ytelse og beskrive omgivelsene elevene arbeider i. Det er her man skal prøve å integrere vanskelige begreper og vise sammenhenger. Den kan benyttes for å sette stemninger og for å tiltrekke oppmerksomhet. Ofte benyttes koder for å forklare f.eks. kartlige data. Disse kodene må forklares på skjermen slik at det er klart hva de forskjellige områdene viser. Nøkkelskjermen skal også gi beskjeder om ting som er relevante for det som skjer. Informasjon om viktige data skal være tilgjemgelig fra nøkkelskjermen, og man skal ha valgmuligheter/menyer og hint om hva som er fornuftig å gjøre i neste omgang. Som man skjønner har nøkkelskjermen mange funksjoner, og å designe slik at alle disse funksjonene ivaretas er ingen enkel oppgave.



Eksempler på slike elementer kan være å vise sammenheng mellom tabelldata og en graf. Dette oppsummerer et forsøk som er gjort, og illustrerer et tallmateriale i grafisk form. Et slikt diagram behøver kanskje ikke ligge og ta opp plass på skjermen hele tiden, men kan tas fram og vises i en

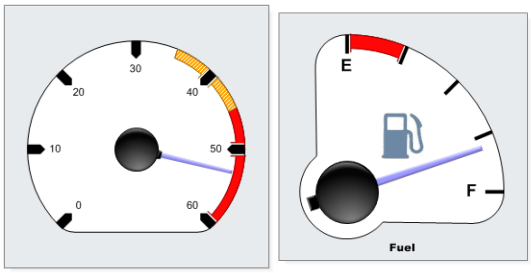
dialogboks. Eller det kan være et poeng å ha denne fremme hele tiden under forsøket slik at tabellen og grafen oppdateres fortløpende. Dette vil kanskje (for noen elever) gi en umiddelbarhet mellom forsøk, tallverdi og graf som vil kunne gi tilleggsinformasjon som de vil føle er nyttig.

Et annet eksempel kan være elementer i nøkkelskjermen som gir feedback på hvor godt eleven har mestret den situasjonen han jobber med. Dette er elementer som måler ytelse. Det kan gjøres på mange måter.



Det er mat igjen for 10 dager...

Dette kan gjøres på mange måter. Som en graf, som en måler eller som en tekstmelding eller lignende.



Ofte vil vi ha behov for å beskrive en "virkelighet" på skjermen. Eksempelet er hentet fra programmet "Swamp Gas goes to Europe" og viser et "unaturlig", men fullt forståelig bilde av Europa med en del hint om hva som ligger hvor, og hvordan man kan komme seg dit. Man ser en "alien" i et romskip som skal navigere rundt på kartet.

I det neste eksempelet prøver man å integrere ideen med koordinater med en posisjon på et kart. Her illustreres at lengde- og breddegrad har noe med en fysisk plassering å gjøre. Man forsøker altså å integrere det abstrakte begrepet med en konkret erfaring - å gi det abstrakte begrepet en forankring i noe konkret.

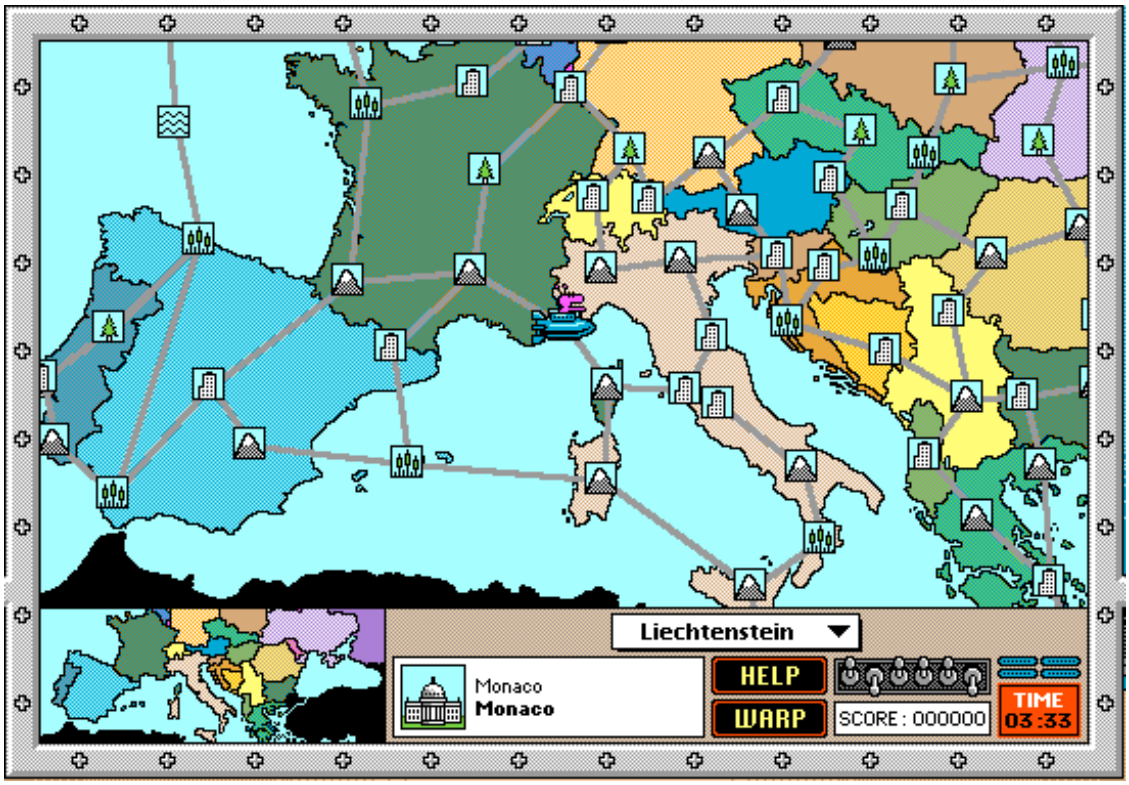
Ofte kan det være lurt å bruke farger og/eller animasjon for å trekke elevens oppmerksomhet mot noe som er viktig. Hvis vi f.eks. lager et program om navigering, vil solens gang over himmelen være viktig. Dette kan da illustreres enkelt med en animasjon.

Som vi skjønner er det mange krav til en nøkkelskjerm. Skjermen er liten og alt det vi ønsker å bruke den til tar plass. Det er selvfølgelig ikke et krav at alt dette

skal inn på alle nøkkelskjermer, men ofte har man et ønske om å få mer inn på nøkkelskjermen enn det egentlig er plass til. Da må man kritisk vurdere alt dette og finne ut hva an er halt avhengig av. Det er ingen god løsning å lage flere skjermbilder. Mye av det vi trenger må ikke nødvendigvis ligge synlig på skjermen hele tinden. Sllike ting kan vi vurdere å legge i egne dialogbokser eller paletter, men det er også viktig å ikke "tapetsere" skjermen med dialogbokser.

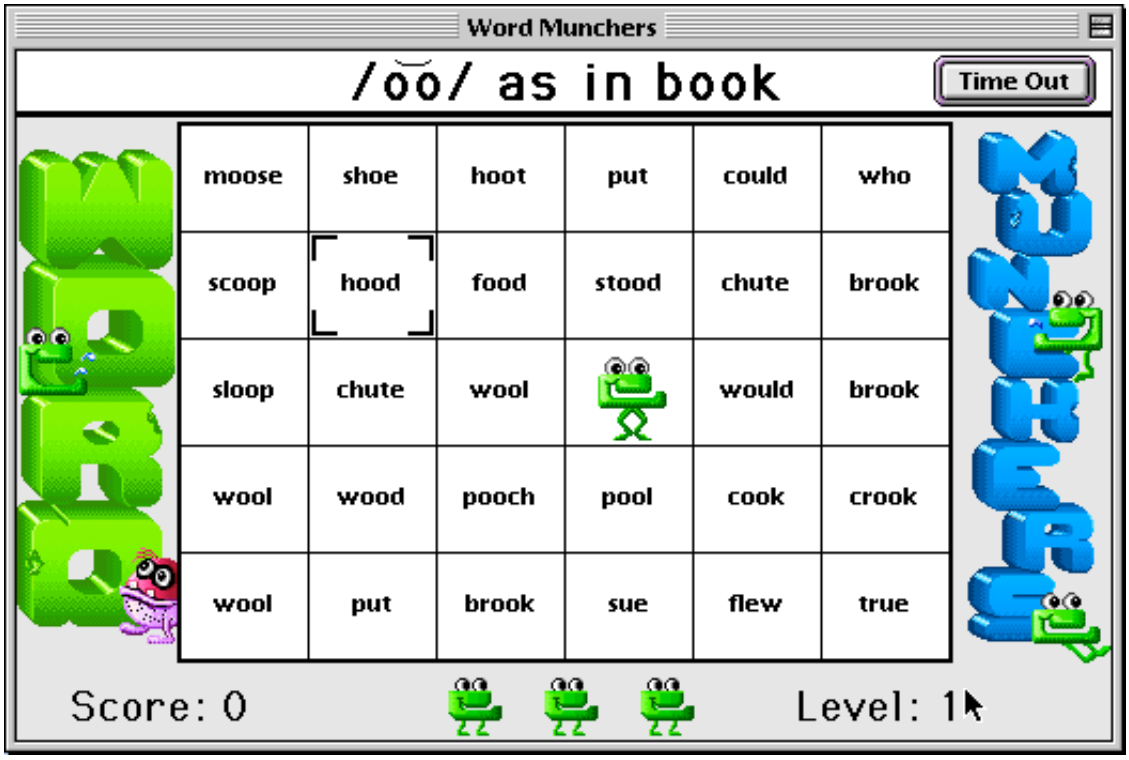
En siste ting man må vurdere å bruke er lyd. Det høres kanskje litt rart ut å snakke om lyd når vi snakker om en nøkkel**skjerm**, men i begrepet nøkkelskjerm ligger alt som naturlig hører inn i det læringsmiljøet vi designer.

Eksempler på nøkkelskjermer:



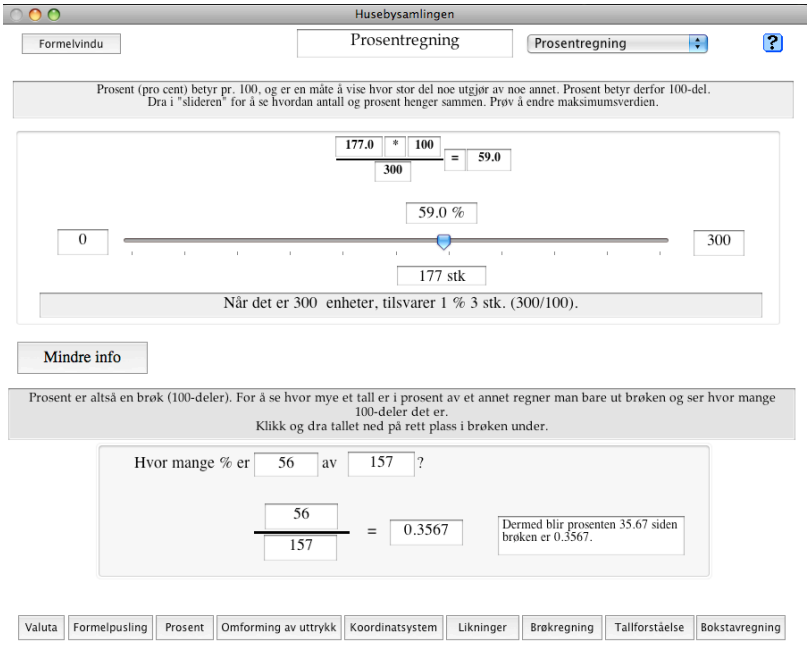
Swamp Gas (http://www.swamp-gas.com) **visits Europe** ble utviklet i 1994. I dette programmet spiller brukeren rollen som det utenomjordiske vesenet Swamp Gas som reiser rundt over Europa og finner ut hvor ting ligger. Man får et sett land man skal besøke. Så må man legge opp en reiserute som man så må gjennom innenfor en gitt tid. På kartet får man hele tiden se alle alternative veier fra der man står i øyeblikket. Hvilket land man skal reise til vises i pop-up menyen rett under kartet. Her kan man velge hvilket land man skal besøke neste gang. De landene man har besøkt er grået ut i menyen. Hver gang man besøker et sted får man tilleggsopplysninger om dette stedet. Man kan spille dette spillet med forskjellige vanskelighetsgrader. Bl.a. kan man gjøre det vanskeligere ved å bruke byer istedenfor land, eller evt. andre landemerker. Man kan også gjøre det vanskeligere ved å gi en kortere tidsfrist.

Word Munchers



Dette var en del av en serie program som ble utviklet av MECC (Minnesota Educational Computer Consortium) på 1980-tallet. Som så ofte er dette bygd over en spillmetafor. I denne versjonen får man oppgaver man må løse under tidspress. Skjermbildet over burde være selvforklarende.

Husebysamlingen



Dette er et program av en litt annen type. Dette er ikke bygd som et spill, men er mer et ekpserimenterende øvingsprogram der man skal finne ut av hvordan man arbeider med prosentregning ved å "utforske" prosentformelen. Her kan man "leke" seg litt med de forskjellige elementene som har en betydning for hvordan man beregner prosentverdier. Man får en liten forklaring samtidig med at man får muligheten til litt mengdetrening.

Interlude

Hittil har vi sett på 2 sider ved et design: hva brukeren skal gjøre og hva programmet skal gjøre som respons på dette. Dette fører til forskjellig fokusering på de forskjellige nivåene i modellen.

Hva eleven skal gjøre:

1. Ut fra erfaringer man har gjort seg (enten som elev eller som lærer), eller på basis av spesielle interesser, har man **valgt et emne**.
2. Deretter har man fokusert på elevene og beskrevet elevenes forutsetninger og hvilke problem man ønsker å løse som en del av en **målsetting**.
3. Så har man valgt en **metafor** som skal hjelpe eleven å være aktiv
4. Dette har så blitt konkretisert i en **aktivitetstabell** som fokuserer på å identifisere hvilke aktiviteter brukeren skal utføre gjennom programkjøringen (elevkolonnen).
    4.1. Aktivitetstabellen skal primært fokusere på de fysiske aktivitetene brukeren skal utføre mens programmet kjører, men det er lurt å tenke på hvilke aktiviteter som også krever en mental aktivitet - det er ikke noe positivt forbundet med "aktiviteter" som bare består av "tanketom knappetrykking".
5. Aktivitetstabellen dukker så opp som boder på **markedet**.
6. som så igjen blir til aktiviteter i **nøkkelskjermen**.
    6.1. En aktivitet i nøkkelskjermen kan være så mangt - det behøver ikke være å velge i en meny. Det kan være å skrive inn koordinater, å ta tak i noe å flytte det, å svare på et spørsmål, å peke på et sted på et kart etc…
    6.2. Pass på at ikke aktivitetene koker ned til å velge i menyer. Det er ofte passivt og kjedelig, og fremmer ikke aktiv læring.

Hva programmet skal gjøre som respons:

1. Første gang vi møter dette er i **metaforen**. Denne gir oss en første idé om hva som vil vises på skjermen.
2. Deretter har man identifisert elementer i denne metaforen som skal hjelpe eleven til å bygge opp en mest mulig korrekt **mental modell** (erfaringsbilde)
3. Dette har så blitt konkretisert i en **aktivitetstabell**, der vi skal lage en kolonne som inneholder det programmet gjør som en respons på elevens aktiviteter (programkolonnen).
4. Erfaringsbildet vil så vises på den sentrale **markedsplassen**.
5. og må også plasseres inn på en fornuftig måte på **nøkkelskjermen**.

Nøkkelskjermsekvens

En nøkkelskjerm er ikke et statisk skjermbilde. Det er i dette skjermbildet (eller vinduet) de spennende tingene skal foregå. Metaforen, aktivitetstabellen, erfaringsbildet og markedsdiagrammet sier hvordan nøkkelskjermen skal se ut, og hvordan den skal endre seg etter hvert som eleven gjør noe under programkjøringen.

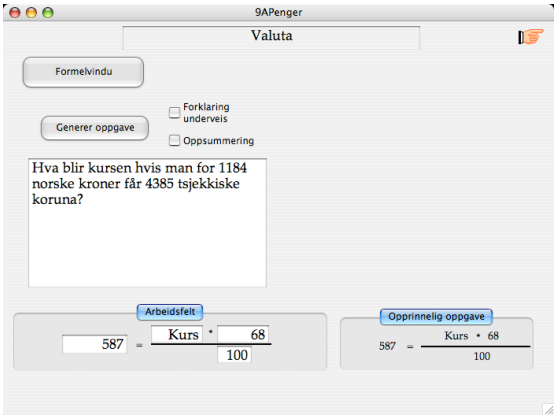
- Vindusinnhold vil endre seg
  - Et fly beveger seg over et kart, kartet ruller
- Meldinger kommer og går
  - Advarsler, meldinger, Hint og hjelp etc...
- Verdier oppdateres
  - Bensinmåler oppdateres, koordinater etyc...
- Kommandoer blir tilgjengelige eller utilgjengelige
- Det skjer animasjoner
  - Flyet beveger seg over kartet, bensinmåleren oppdateres fortløpende etc…

Alt dette skjer på en nøkkelskjerm. Det er svært mange dynamiske elementer som må tas hensyn til når man designer nøkkelskjermen. Vi må vurdere alle detaljene i designet. Hva skjer på skjermen, og hvordan forandres skjermen når eleven gjør noe? Vi må ta stilling til alle mulige aksjoner fra elevens side, både de fornuftige og de ufornuftige. Vi må også ta hensyn til aksjoner fra elever som vil prøve å få programmet til å oppføre seg ufornuftig. Vi må lage nøkkelskjermsekvenser - dvs. spesifisere de forskjellige tilstandene nøkkelskjermen kan befinne seg i. Det er ikke snakk om nye skjermbilder, men nye instanser av det "samme" skjermbildet. Med dette menes utgaver av skjermbildet der noen elementer er blitt ednret. Vi vil da finne pedagogiske spørsmål som må avklares. Kanskje må vi tilbake og revurdere deler av formgivingen som vi trodde vi var ferdige med.

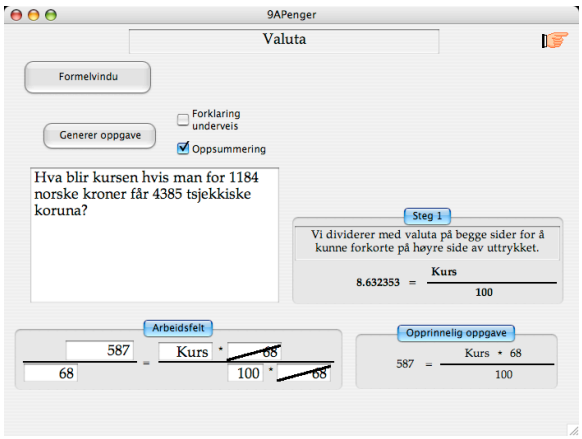
Eksempel - Husebysamlingen

Utgangsposisjon. Dette er valuta-delen av programmet. Hva skjer når eleven

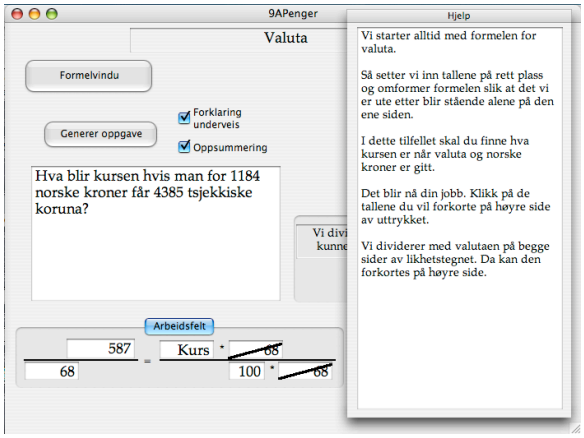
- klikker i oppgavefeltet
- klikker på et av feltene i arbeidsfeltet
- klikker på en av knappene
- …



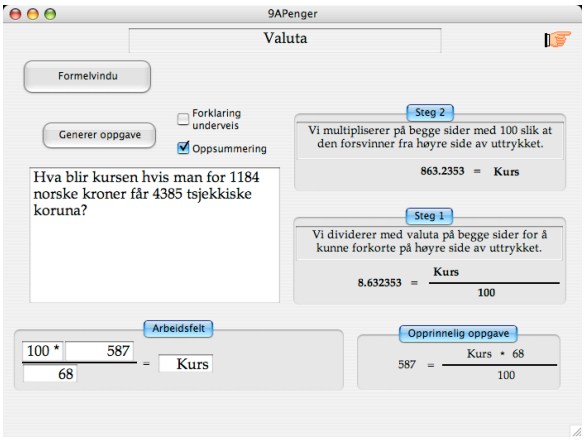
Her har eleven klikket på tallet 68 i arbeidsfeltet. Da dukker et felt med 68 opp i nevneren på begge sider i likningen. Etter en kort stund settes det en strek over 68 i både teller og nevner på høyre sude av likningen og 68 forsvinner (forkortes). Fordi eleven har krysset av for "Oppsummering", dukker feltet "Steg 1" opp på høyre side av vinduet. Her er utregningen av venstre side gjort, slik at eleven kan se resultatet av det han har gjort.



I neste skjermbilde har eleven krysset av for "Forklaring underveis". Da dukker det opp en palett (som alltid ligger øverst på skjermen) som forklarer hva man skal gjøre og hva resultatet av elevens aktivitet er. Paletten forsvinner når vi fjerner avkryssingen for "Forklaring underveis".



Denne figuren viser tilstanden av skjermbildet etter at eleven har trykket på 100-tallet i nevneren i arbeidsfeltet. Dermed skjer forkortningen som er vist i bilde 1. Siden eleven gjorde det etter at han hadde forkortet det første tallet, og eleven har krysset av for "Oppsummering", dukker det opp et felt "Steg 2" til høyre som viser resultatet av beregningen.



Og så videre med alle mulige tilstander av skjermen. Det er viktig at man tenker gjennom all interaksjon i skjermbildet og tegner/viser hvordan denne vil endre seg underveis for å forsikre oss om at vi har tenkt gjennom alle sider ved designet. Som man kan se jobber vi hele tiden på den samme skjermen/i det samme vinduet, men dette er dynamisk og endrer seg etter hvert som vi utfører noen aktiviteter der.

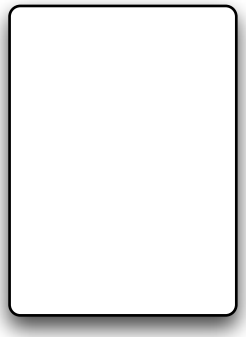
Tilstandsdiagram

Det neste steget i metoden er, på en kompakt måte, å beskrive dynamikken i designet. Det er nødvendig å ha en oversiktlig og nøyaktig metode for å beskrive hvordan skjermbildet endrer seg ettersom programmet blir brukt. Dette er nødvendig av to hensyn:

1. Kommunikasjon med programmerer. Ofte blir et design satt bort til implementasjon av en tredjepart, og da blir det nødvendig å ha en eksakt måte å beskrive interaksjonen på. Man kan ikke forvente å ha løpende kontakt med programmerer, og hvis man har uklarheter i designet vil programmereren alltid velge den løsningen som medfører minst arbeid. Noe av dette kan avhjelpes gjennom prototyping der man ved designet kan legge en prototype som viser klart hva man har tenkt.

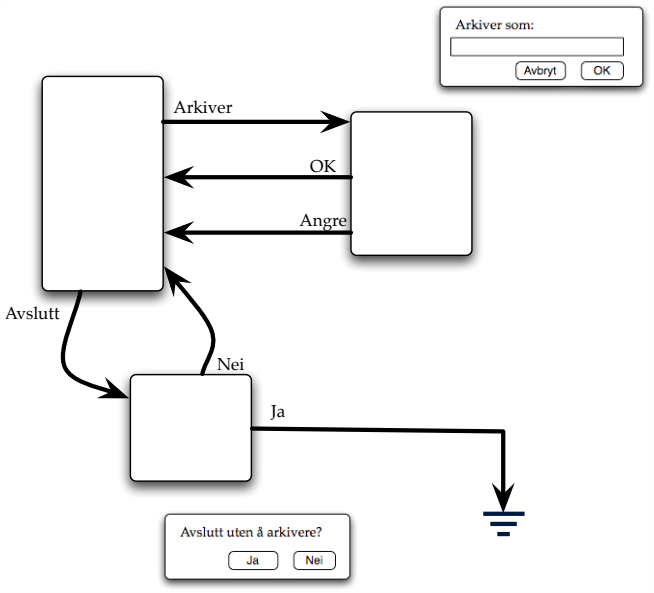
2. Avsløre konsistensproblemer i programmet. Selv om vi har gjennomført designet så godt vi kan, kan det fremdeles foreligge inkonsistenser i det - det kan være ting vi ikke har tenkt på. Gjennom å beskrive interaksjonen eksakt i et tilstandsdiagram vil man kunne avsløre slike.

TIl dette bruker vi et tilstandsdiagram. Med dette menes i denne sammenhengen et diagram som viser endringene i skjermbildets tilstand etter som programmet blir brukt. Man knytter ikke verdier av variabler, objekter eller annet til tilstandsdiagrammet. En tilstand markeres som et rektangel med avrundete hjørner.



Inn og ut av disse tilstandene går det piler. Når en pil peker ut av en tilstand skal det der pilen går ut beskrives hvilken handling som førte til tilstandsendringen.

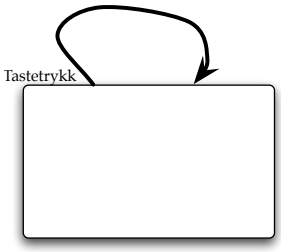
Et enkelt eksempel:



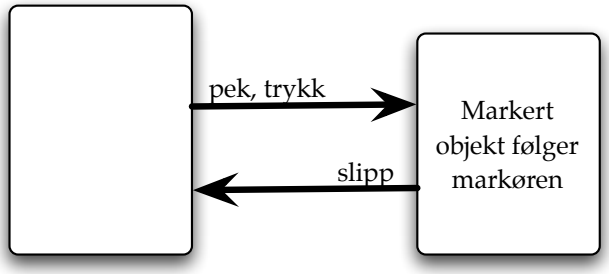
Dette viser hvordan man kan tegne inn en arkiver/avslutt sekvens i et tilstandsdiagram. Som et tillegg i tilstandsdiagrammet er det skissert hvordan "under-tilstandene" ser ut (modale dialogbokser).

Ved å be om å arkivere skjer det en endring i skjermbildet. Dette markeres som en pil ut av hovedtilstanden. Siden

dette valget fører til at det åpnes en dialogboks, havner vi i en annen tilstand. Hvordan denne ser ut er vist som en tegning i tilstandsdiagrammet. En annen måte å gjøre dette på ville vært å gitt tilstanden et nummer og så beskrive utseende av tilstanden på et annet ark. Man kan også godt ha en pil som går ut av en tilstand og som så kommer tilbake til den samme tilstanden. Dette vil f.eks. gjøres hvis man skal ha mulighet til å skrive på tastaturet.

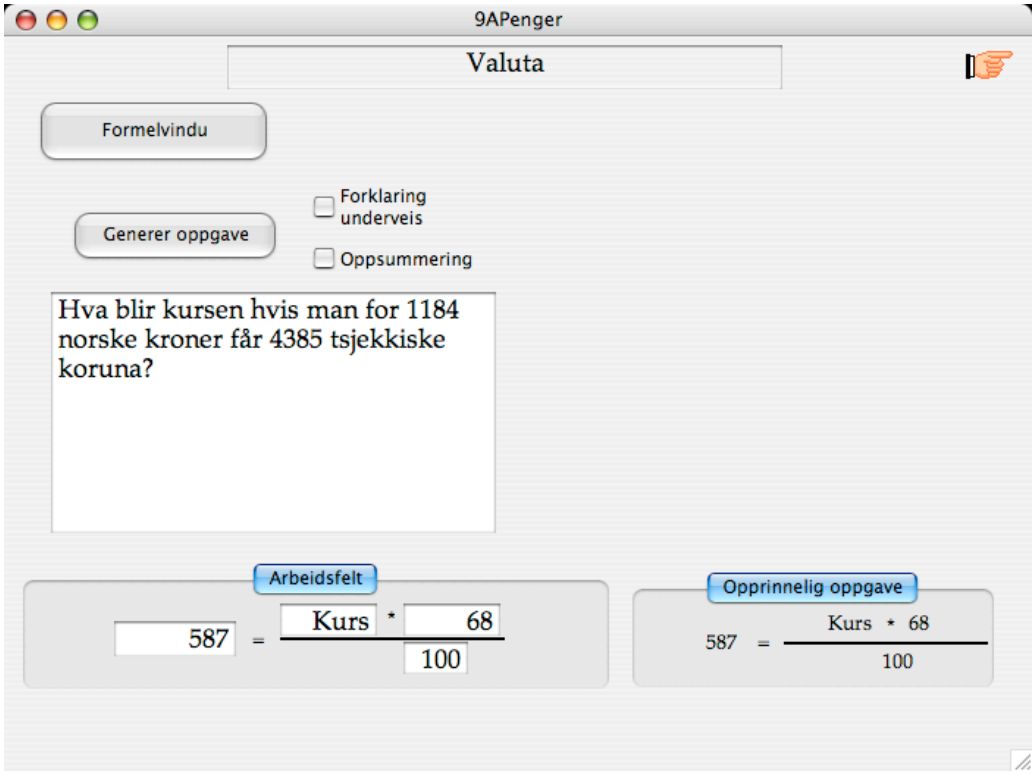


I andre tilfeller kan tilstandsendringene skje gjennom aksjoner som ikke er direkte synlige på skjermen. Eksempler på dette kan være direkte manipulerte dialoger der man kan ta tak i et objekt og flytte eller endre dem. Dette fører til en endring i en tilstand. Så lenge man drar i objektet vil man være i en annen tilstand fordi man ikke vil kunne gjøre noe annet så lenge museknappen er nede. Dette kan markeres på denne måten:

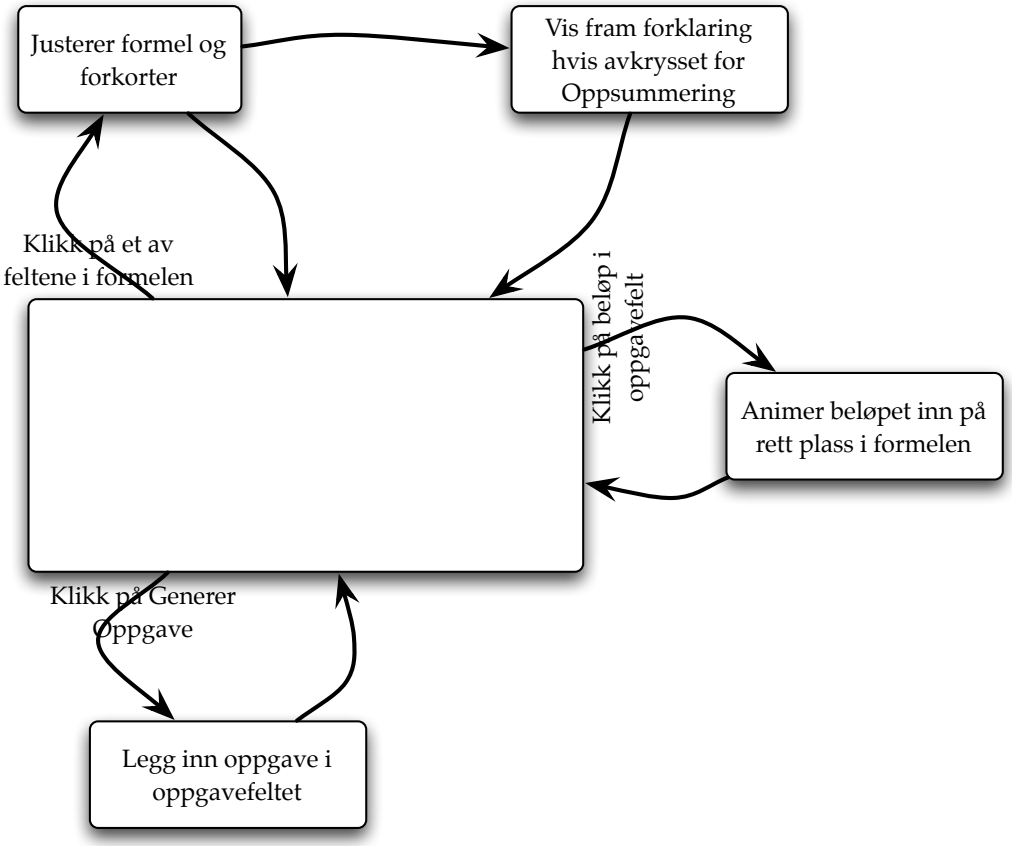


Eksempel fra Husebysamlingen:

Skjermbilde:



Tilstandsdiagram:



Dette er et enkelt eksempel som ikke dekker hele funksjonaliteten i skjermbildet på forrige side. Et fullstendig tilstandsdiagram for et program kan bli ganske stort og komplekst. Da bør man kanskje bryte det ned i mindre bestanddeler som blir mer håndterbare.