NTNU
Det skapende universitet

Institutt for Datateknikk og Informasjonsvitenskap

Løsningsforslag
(Løsningene i kursiv)

Eksamensoppgave i TDT4180 Menneske-Maskin-Interaksjon (MMI)

Faglig kontakt under eksamen: Dag Svanæs

Tlf.: 91897536

Eksamensdato: 27. mai

Eksamenstid (fra-til): 0900 - 1300

Hjelpemiddelkode/Tillatte hjelpemidler: D / Ingen trykte eller håndskrevne hjelpemidler tillatt. Bestemt enkel kalkulator tillatt.

Annen informasjon:

Målform/språk: Bokmål

Antall sider: 9 (inklusive forsiden)

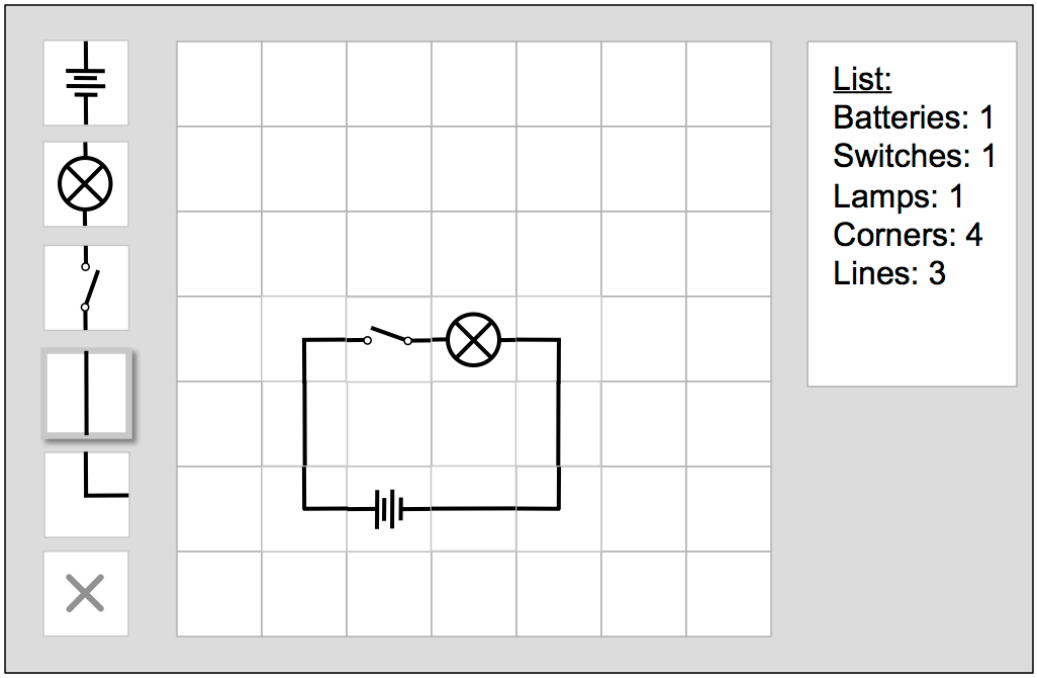
Antall sider vedlegg: 0

Kontrollert av:

_____

Dato Sign

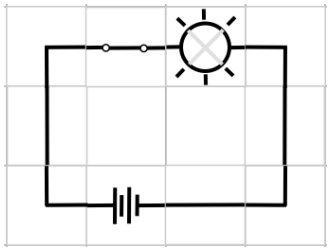
Oppgave 1 Grensesnittdesign (30%)



Figuren over viser skisse til programmet STRØM som skal lære barn om elektrisitet. Til venstre finnes en rekke med ikoner: fem byggestener og et sletteikon. I midten er et 7x7 rutenett der brukeren kan bygge elektriske kretser. Til høyre er en liste med de byggestenene som til enhver tid er brukt. Listen oppdateres automatisk når brukeren bygger kretsene. Programmet skal kunne kjøres som en Java-applet i en nettleser.

Programmets virkemåte er som følger:

- Brukeren velger aktivt ikon ved å trykke med venstre museknapp i ikonlisten. I eksempelet over er rett linje (ledning) valgt som aktivt ikon.
- Brukeren kan rotere aktivt ikon i ikonlisten 90 grader mot høyre ved å klikke med høyre museknapp i ikonet.
- Brukeren kan plassere en kopi av aktivt ikon (med valgt rotasjon) i rutenettet ved å trykke med venstre museknapp i en rute i rutenettet.
- Dersom valgt ikon er sletteikonet så vil et klikk i en rute i rutenettet føre til at innholdet i denne ruten slettes og ruten blir hvit.
- Et bryterikoner kan endres fram og tilbake mellom åpen og lukket tilstand ved å trykke på den med høyre museknapp.
- En lampe er enten av eller på (lyser) avhengig av om det går strøm i den eller ikke. Figuren under viser et utsnitt fra eksempelet over der bryteren er lukket, noe som fører til at det går strøm i kretsen og lampen lyser.



a) Diskuter brukervennligheten til den foreslåtte virkemåten med utgangspunkt i begrepene ”affordance” og konsistens (”consistency”).

”Affordance” er et begrep som ble introdusert til MMI-faget gjennom Don Norman. Det kommer opprinnelig fra en del av psykologen som heter ”Økologisk psykologi” (J.J. Gibson). Slik Norman beskriver affordance så er det hva et produkt eller en del av et produkt signaliserer om sin bruk gjennom sin utforming. For eksempel så signaliserer Colaflasken gjennom sin form at den kan gripes og drikkes av.
I designet over så handler affordance om grensesnittets evne til å formidle hva brukeren kan gjøre og hva som blir konsekvensene.

- *Ikonene til venstre er ikke utformet som knapper. Det er derfor ikke åpenbart at de skal trykkes på.*
- *Det er ikke åpenbart hva som skal skje når man trykker på et ikon.*
- *Det er ikke klart at ikonene ikke kan dras ut på rutenettet med ”drag&drop”. Mange med erfaring fra touch-grensesnitt vil antagleig forvente dette, spesielt fordi ikonene ikke er utformet som knapper.*
- *At ikonene kan roteres med høyre-klikk er ikke på noen måte åpenbart. Det er ingenting som signaliserer dette. Eneste måten å eventuelt finne ut dette på er ved å prøve seg fram.*
- *Det er kanskje heller ikke åpenbart at krysset betyr ”Sletteikon”. Dette er ikke en byggesten som de andre ikonene, men en funksjon. Det kommer ikke frampå noen måte.*
- *At bryteren kan slås av og på med høyreklikk når den er plassert ut i rutenettet er heller ikke åpenbart.*
- *I listen over elementer så er ordet ”<u>List:</u>” understreket. Plassert i en ramme i en nettleser så vil dette lett kunne tolkes som en link som brukeren kan trykke på, noe det i dette tilfelle ikke er.*

”Consistency” handler om at ting skal se og oppføre seg likt i forskjellige deler av et grensesnitt. I forhold til den plattformen som applikasjonen kjører på så handler det om at for eksempel knapper i en Windows 7 applikasjon skal se ut som knappene i Windows 7. Internt i applikasjonen så handler det om at ting skal se og oppføre seg likt i forskjellige deler av applikasjonen. Dette gjelder også begrepsbruk. For eksempel så er det mangel på konsistens dersom noe heter ”dokument” et sted og ”fil” et annet sted i applikasjonen.
I designet over så er det en del eksempler på dårlig konsistens:

- *Ikonene til venstre er egentlig knapper, men de ligner ikke på knapper, verken i Swing, Mac OSX, Windows eller annet.*
- *Den understrekne ”List” teksten er ikke konsistent med hvordan tekst brukes i en nettleser, der understreking må unngås for ikke å forlede brukeren til å tro at det er en link.*
- *Høyreklikk i ikonlisten betyr rotasjon, mens høyreklikk i rutenett kun virker på brytere, og da med en helt annen funksjon (av/på).*
- *Listen til høyre er sortert i annen rekkefølge enn listen av ikoner (en liten detalj).*

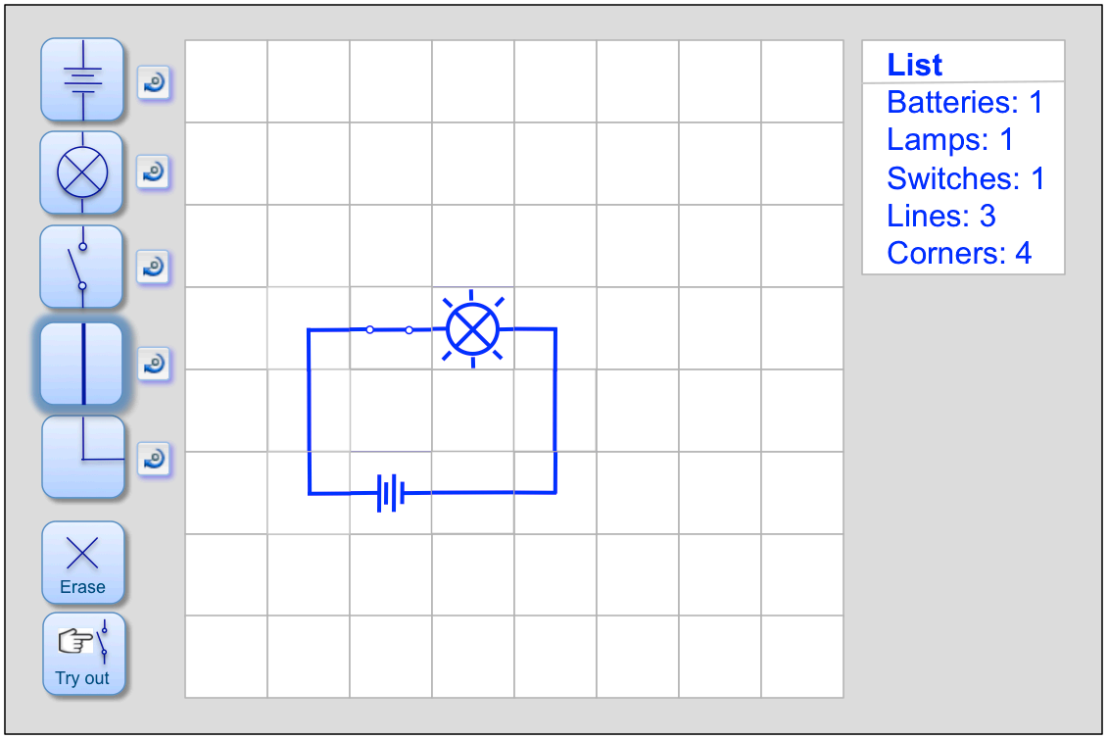
Karakter 1a:

A. *God forklaring av begrepene og god analyse. Behøver ikke ha med alt fra løsningsforslaget, men det viktigste.*
B. *Mangler litt på teori og analyse, men alt som skrives skal være riktig.*
C. *Relativt lite teori, og en del mangler i analyse.*
D. *Mangler teori, og store mangler i analyse.*
E. *Store mangler.*
F. *Ikke besvart, eller ikke relevant svar.*

b) Basert på diskusjonen fra oppgave 1a, foreslå et alternativt brukergrensesnitt som du mener vil være bedre. Skisser løsningen og begrunn svaret.

Analysen i 1a avdekket en del konkrete problemer som antagelig kan løses med forbedringer i utformingen av brukergrensesnittet.

1. *Ikonene til venstre bør de ut som knapper. Dette kan gjøres ved å benytte standard knapper i et rammeverk som Swing, eller lage knapper som ser ut som knapper i HTML eller på PC/Mac. Man kan også velge å bruke andre knapper hentet fra for eksempel iPhone eller Android.*
2. *Det bør være tydelig at dette er en knapperekke, der det finnes en aktiv knapp. Aktiv knapp er indikert i skissen over, men den kan gjøres klarere. Man kan benytte standard i for eksempel Swing, OSX, Windows el.l., eller man kan velge en sterkere markering som for eksempel invertering og stor ramme.*
3. *Det at det ikke er et drag&drop-grensesnitt blir forhåpentligvis tydelig ved å gjøre ikonene til knapper. Detteer noe av det som må verifiseres ved brukebarhetstesting.*
4. *Rotasjon med høyreklikk er ikke intuitivt. En mulig løsning er å legge på en hjelpetekst "Høyreklikk for å rotere". En annen løsning, som nok er bedre, er å lage små rotasjonsknapper til høyre for de knappene som kan roteres.*
5. *At krysset betyr "sletteikon" kan gjøres mer tydelig ved å skape fysisk avstand mellom dette ikonet og resten av ikonlisten. Den vil da framstå som noe annet enn en byggesten (gestaltprinsippene). Her kunne det kanskje også vært en tekst "Slett" / "Erase" i ikonet eller under det.*
6. *Høyreklikk for å slå av/på bryter fungerer dårlig. En hjelpetekst er mulig. En annen løsning er å lage en egen knapp som fungerer som et verktøy for å slå av og på brytere. Det ville da være et funksjonsikon i tillegg til "Erase". Kunne for eksempel være bilde av en hånd som slår på en bryter. Kunne ha følgetekst "Prøv ut"/"Try out".*
7. *Understrekning i "List" bør fjernes.*
8. *Knappene bør ligne knapper på plattformen det kjøres på. For eksempel kan man bruke HTML5 eller Swing sine knapper.*
9. *Gjør rekkefølgen i listen til høyre lik rekkefølgen på ikonene.*



I skissen over er forbedringsforslagene samlet i ett design. Vi ser at ved å sette på "rotasjonsikoner" ved byggestenene så fikk vi "gratis" at det skapes en visuell forskjell mellom de fem byggestenene og de to funksjonsikonene "Slett" og "Prøv ut".

For å øke konsistensen så kunne det også ha vært mulig å rotere ikoner etter at de er utplassert. Det kunne gjøres v.h.a. et tredje funksjonsikon ("Roter"/"Rotate"). Et slikt ikon ville kanskje ha skapt forventning om at det også kunne brukes til å rotere ikonene i lista til venstre, noe som ikke er ønskelig.

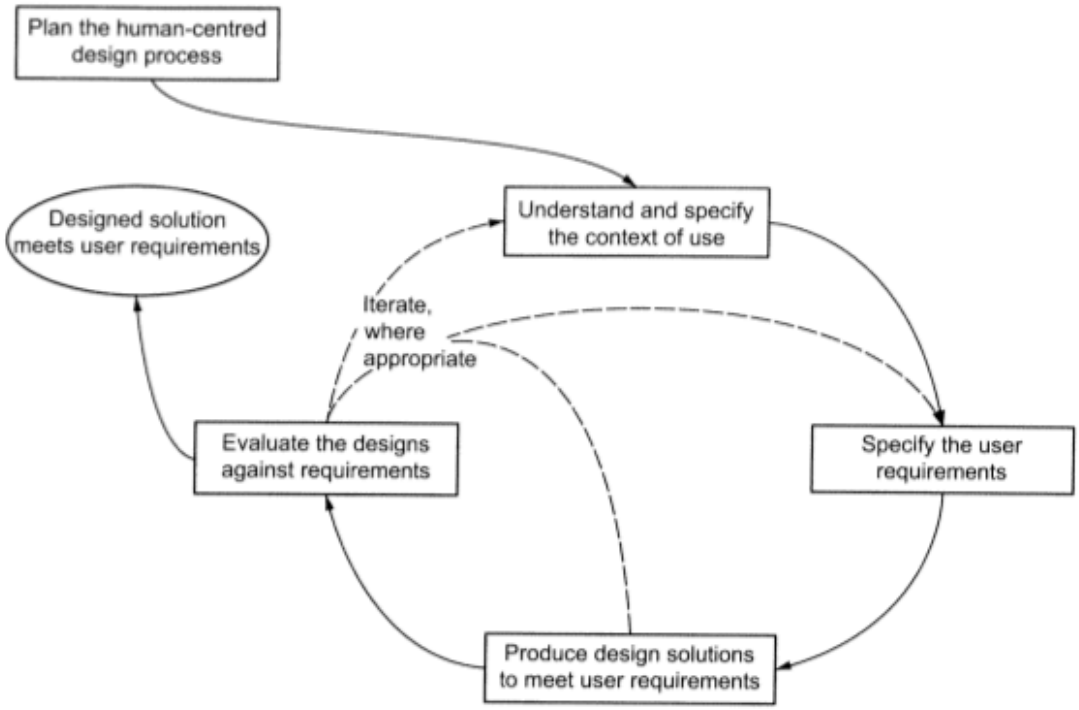
Karakter 1b:

- *A. Gode forslag til redesign basert på analysen i 1a.*
- *B. En del mangler, og noe manglende referanse til 1a.*
- *C. En del gode forslag, men med referanser til analysen i 1a.*
- *D. Mangler i forslag og referanse*
- *E. Store mangler.*
- *F. Ikke besvart, eller ikke relevant svar.*

På 1b trekker det opp litt dersom kandidaten også har kommet med andre forbedringsforslag, selv om det ikke spørres om dette.

Oppgave 2. Brukersentrert design (30%)

ISO standarden 9241-210 (2010) angir fasene i en brukersentrert systemutviklings-prosess som vist i figuren under:



Programmet STRØM er foreløpig kun på et veldig tidlig skissestadium, og man har ennå ikke involvert brukergruppen i designprosessen. Det er vedtatt å kjøre prosjektet som en brukersentrert prosess.

a) Angi to aktuelle brukersentrerte aktiviteter for hver av de fire stegene i den iterative prosessen i ISO-standarden, og argumenter for hvorfor disse åtte aktivitetene er aktuelle for prosjektet STRØM.

1. *Forstå brukskontekst. Her er det viktig å forstå brukeren, de oppgavene som skal utføres, og den omgivelsen det skal skje i. Aktuelle aktiviteter:*
    a. *Feltstudie – Det kunne for eksempel være å ta en tur til en skole der de underviser i O-fag om elektrisitet. Sitte i klasserommet en time og studere hvordan dette blir undervist og hvordan elevene handterer det.*
    b. *Intervju – Aktuelle intervjuobjekter er lærere, elever og skolebokforfattere. I forkant av slike intervjuer er det lurt å lage en intervjuguide med en liste av de temaene man har tenkt å gå igjennom rundt brukskontekst.*
2. *Kravinnhenting. Her er det viktig å få på plass hvilke overordnede krav som skal stilles til en slik løsning. Skal den for eksempel kunne brukes uten at læreren er tilstede? Hvilken aldersgruppe skal den være for? Skal den stå på egne ben, eller være del av et større læringsopplegg,,,,.? Aktiviteter:*
    a. *Intervjuer – Igjen vil det være intervjuer, men nå med fokus på krav. Det handler om å gjøre noen valg – og å skaffe empirisk støtte for disse valgene.*

   b. *Fokusgruppe – Skissen som er laget kan danne utgangspunkt for en fokusgruppe. Man samler da for eksempel fem lærere og starter en diskusjon. Igjen så bør man ha en intervjuguide med de temaene en ønsker å gå igjennom.*
3. *Utvikle designløsninger. Det er her viktig å bruke metoder som gjør det lett å kommunisere med brukere og kunder v.h.a. konkrete eksempler på hvordan systemet skal brukes. Det er viktig å ikke bruke mer energi på en prototyp en strengt tatt nødvendig ("Just enough prototyping"). Aktiviteter:*
   a. *Papirprototyper – Ved å bruke enkle løsninger på papir kan man få i gang en dialog om løsningen. Sammen med et tilstandsdiagram så kan en slik prototyp også brukes til å gjøre s.k. "Wizard of Oz" brukbarhetstester.*
   b. *Kjørende prototyper. Disse kan lages i for eksempel Swing, men kan kan også bruke enkle "wireframe" verktøy eller Powerpoint.*
4. *Evaluere løsninger med brukere. Det er her viktig å få tilbakemeldinger fra brukere som kan brukes videre i prosessen med redesign. Aktiviteter:*
   a. *Papirprototyper kan testes med Wizard-of-Oz brukbarhetstester. Her tar en person rollen som datamaskin og oppdaterer skjermbildet med forskjellige papirlapper avhengig av hva brukeren gjør. Kjørende prototyper kan evalueres med brukbarhetstester på PC.*
   b. *Fokusgrupper. Prototyper kan også evalueres i fokusgrupper der flere brukere/kunder samles for å gi tilbakemelding og diskutere.*

Karakter 2a:

A. Gode forslag to aktiviteter på alle 4 steg. Med nok tekst til at det viser forståelse utover opplisting.
B. Noen mangler.
C. Riktige aktiviteter, men med en del mangler i forklaring.
D. Store mangler
E. Veldig store mangler.
F. Ikke besvart, eller ikke relevant svar.

b) ISO-standarden angir at man skal sette sammen tverrfaglige team. Hvilke kompetanser ville du knytte til deg dersom du var prosjektleder for STRØM? Begrunn svaret.

Med tverrfaglighet menes at man skal ha kompetanser som dekker de forskjellige aktivitetene som skal gjøres i prosjektet. I forhold til dette prosjektet så ser jeg følgende kompetanser som sentrale:

- *Programmeringskompetanse – Det er viktig å ha folk på teamet som er i stand til å utføre implementasjonen av løsningen. Det er også viktig at programmererne er med tidlig i prosjektet for å kunne gi tilbakemelding på fordeler og ulemper med en del plattform og arkitekturvalg.*
- *Testing- og brukerkontaktkompetanse – Det skal utføres tester med brukere, samt fokusgrupper og intervjuer. Det er da nødvendig med noen som har kompetanse på dette, og som spesielt har erfaring med testing, intervjuer og fokusgrupper med barn og ungdom.*

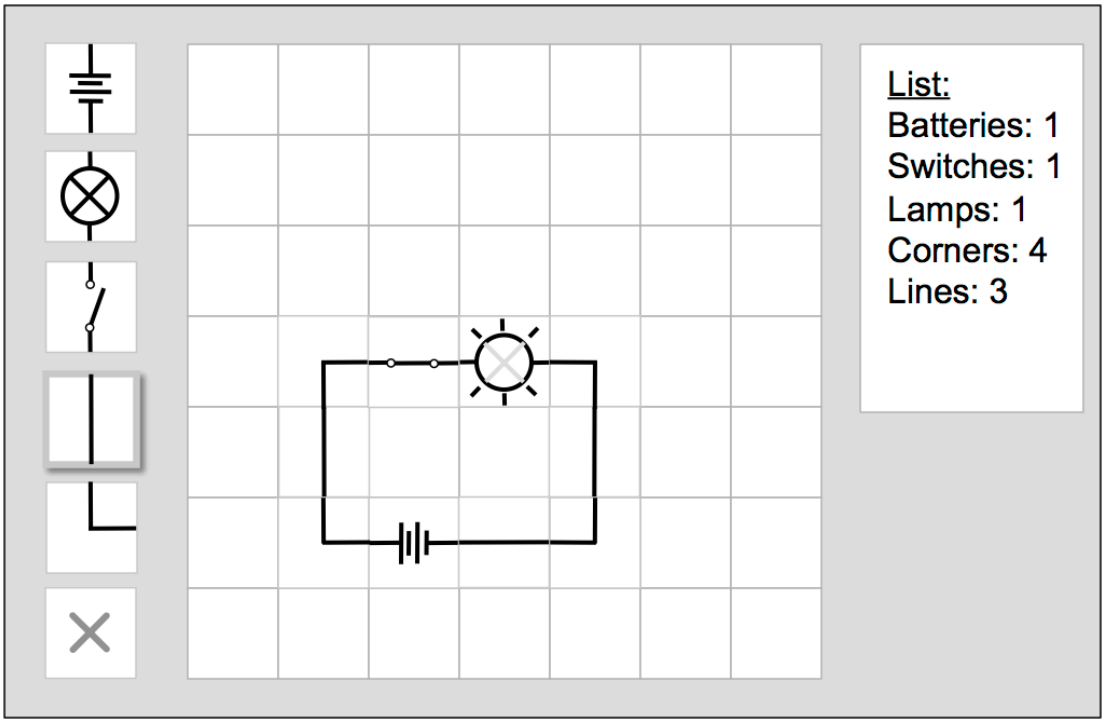
- *Designkompetanse – Det skal lages brukergrensesnitt som passer for målgruppen (barn og ungdom), både i forhold til grafisk design og design av brukerdialogen. Dette krever en designer med kunnskap om digitale medier – en interaksjonsdesigner.*
- *Pedagogisk kompetanse – Appen STRØM skal brukes til å lære om elektrisitet. Det er da viktig å ha kompetanse på læring. Hva skal til for at programvare kan fungere pedagogisk?*

I noen tilfeller vil en og samme person kunne ha flere kompetanser, for eksempel interaksjonsdesignere som er dyktige på å arbeide med barn og unge. Det er også viktig at de som jobber på prosjektet har erfaring fra å jobbe på tverrfaglige team, slik at de forstår sin egen kompetanse og hvordan den kan spille sammen med andres.

Karakter 2b:

A. *Gode forslag til kompetanser. Skal minst dekke det som er foreslått over. Med nok tekst til at det viser forståelse utover opplisting.*
B. *Noen mangler.*
C. *I hvertfall testing/brukerkontakt og design, men med en del mangler i forklaring.*
D. *Store mangler*
E. *Veldig store mangler.*
F. *Ikke besvart, eller ikke relevant svar.*

Oppgave 3. Brukergrensesnittkonstruksjon (40%)



Skissen til programmet STRØM i oppgave 1 (figuren over) skal implementeres i JAVA/SWING.

a) Hvordan vil du bruke Model-View-Controller (MVC) til å skille presentasjon og data i din løsning av dette programmet?

Jeg vil la en felles modell holde informasjonen om den elektriske kretsen som bygges. Det vil si hvilke byggestener som er på hvilken plass i matrisen, og deres rotasjon. I tillegg så vil modellen kunne beregne statistikken over antall brukte byggestener som brukes i vinduet til høyre.
Jeg vil så ha et arbeidsområde (WorkAreaView) som holder de 7x7 elementene (CircuitElement), og et statistikkvindu (ListView) som viser statistikken.
Når brukeren legger til eller fjerner et element i arbeidsområdet, så vil modellen oppdateres og endringsmelding sendes bl.a. til statistikkvinduet som vil oppdatere statistikken.

b) Forklar hovedtrekkene i din løsning i forhold til valg av modell/modeller.

Jeg ville bare ha en modell, den for å holde på den elektriske kretsen. Man kunne også tenke seg å ha en modell som holdt aktiv byggesten/funksjon, men det virker litt overflødig.
Jeg velger å bruke den ferdige klassen PropertyChangeSupport for å handtere listen av lyttere.

Jeg velger å la de faktiske grafiske ikonene for byggestenene i fire rotasjoner ligge i hovedvinduet (MainPanel). De kan hentes derfra med metoden:

- *Icon getIcon(int type, int rotation)*

c) Beskriv modellen/modellene i detalj.

Modellen vi måtte holde informasjon om hva som befinner seg på de enkelte posisjonene i matrisen. For hvert element trenger den byggestentype og rotasjon. Den enkleste løsningen er å legge definisjonen av byggestenene i modellen som statiske variable (public static final) for de 5 byggestenene. For eksempel BATTERY = 1, LAMP = 2 etc., og en egen verdi for tomt element (EMPTY = 0). Rotasjonen kan uttrykkes på samme måte som et tall mellom 0 og 3 for h.h.v. 0, 90, 180 og 270 graders rotasjon.

For å lese og skrive elementer så trenger vi:

- *int getElementType(int x, int y) - Hent elementtypen på posisjon x,y.*
- *int getElementRotation(int x, int y) - Hent rotasjonen på posisjon x,y.*
- *void setElement(int x, int y, int type, int rotation) - Sett elementtyp og rotasjon på posisjon x,y.*

For å hente ut statistikk så trenger vi metoder for hver av de fem typene. Dette kan gjøres med en metode, der byggestentype sendes inn som parameter:

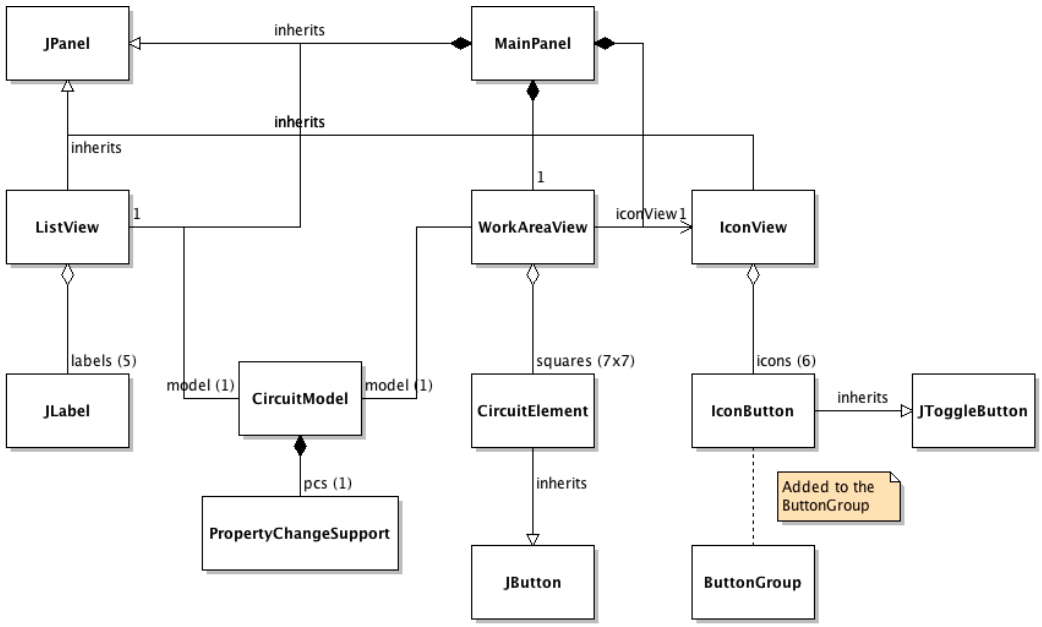
- *int getElementCount(int type)*

I tillegg så må objekter (views) kunne legge seg til som lyttere til modellen. Det gjør de ved å kalle følgende metode i modellen:

- *void addPropertyChangeListener(PropertyChangeListener listener)*

Modellen vil videresende dette kallet til sitt PropertyChangeSupport-objekt.

d) Vis klassediagrammet for løsningen din.



Vi ser i figuren at:

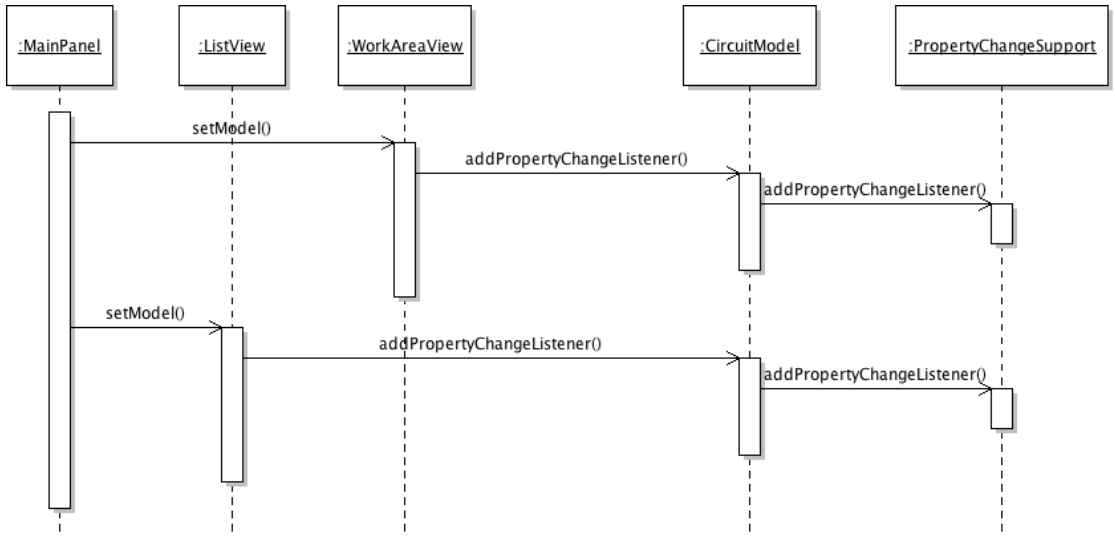
- *Klassene MainPanel, ListView, WorkAreaView og IconView alle arver fra JPanel.*
- *MainPanel-objektet har ett ListView-objekt, ett WorkAreaView-objekt og ett IconView-objekt.*
- *CircuitModel har ett PropertyChangeSupport-objekt for å holde på sin liste av lyttere.*
- *Både ListView og WorkAreaView har en referanse til sin modell (CircuitModel).*
- *ListView-objektet har 5 JLabel-objekter.*
- *WorkAreaView-objektet har 49 CircuitElement-objekter (7x7 matrise).*
- *CircuitElement arver fra JButton.*
- *IconView-objektet har 6 BuildingBlock objekter.*
- *WorkAreaView-objektet har en referanse til IconView-objektet (for å vite hvilket ikon som er aktivt).*
- *IconView-objektet har 6 IconButton-objekter (de 5 byggestenene + erase ikonet)*
- *IconButton arver fra JToggleButton.*
- *Alle de 6 IconButton-objektene tilhører en ButtonGroup (er blitt lagt til ved add()).*

I tillegg:

- *Både IconButton og CircuitElement må implementere MouseListener for å kunne plukke opp museklikk med høyre museknapp.*
- *Både IconButton og CircuitElement må også kunne ta imot sine egne ActionEvents. De må følgelig også implementere ActionListener.*

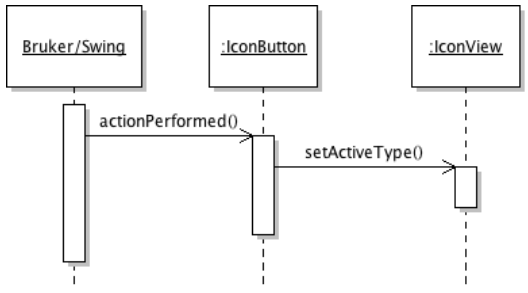
- *Både WorkAreaView og IconView vil trenger å kunne referere sin overordnede JPanel (MainPanel), og må følgelig ha en "tilbakereferanse" dit.*
- *På samme måte så vil de 6 IconButton-objektene måtte ha en "tilbakereferanse" til sitt IconView, og de 7x7 CircuitElement-objektene ha en " tilbakereferanse" til sitt WorkAreaView.*

e) Tegn opp sekvensdiagrammet når det hele startes opp. Forklar.



Sekvensdiagrammet over viser hvordan de to view-objektene kobles til sin modell.
I tillegg så må alle de 6 ikonene og de 49 elementene legge til seg selv som lytter, både på action events og på mouse events.
WorkAreaView må få vite om sitt IconView, og alle ikoner og elementer må når de skapes få en referanse til det view som eier de.
I tillegg så må alle IconButton-objekter legges til en ButtonGroup slik at de blir del av en gruppe som får radio button oppførsel.

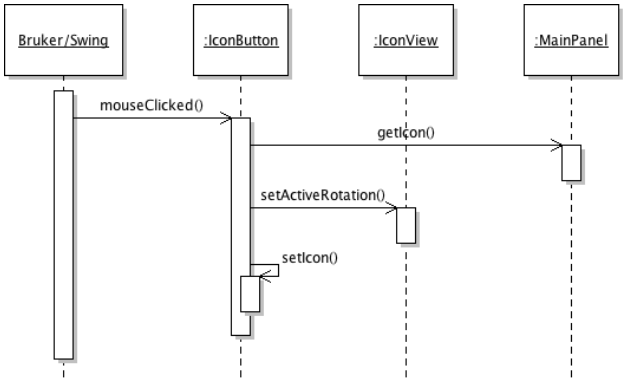
f) Tegn opp sekvensdiagrammet når brukeren velger et nytt aktivt ikon.



I sekvensdiagrammet over ser vi at brukeren klikker på et ikon. Det fører til at Swing sender meldingen actionPerformed() til ikonknappen. Alle ikonknapper vet hvilken elementtype de har (satt ved oppstart), og ikonknappen oppdaterer denne informasjonen i sitt IconView.

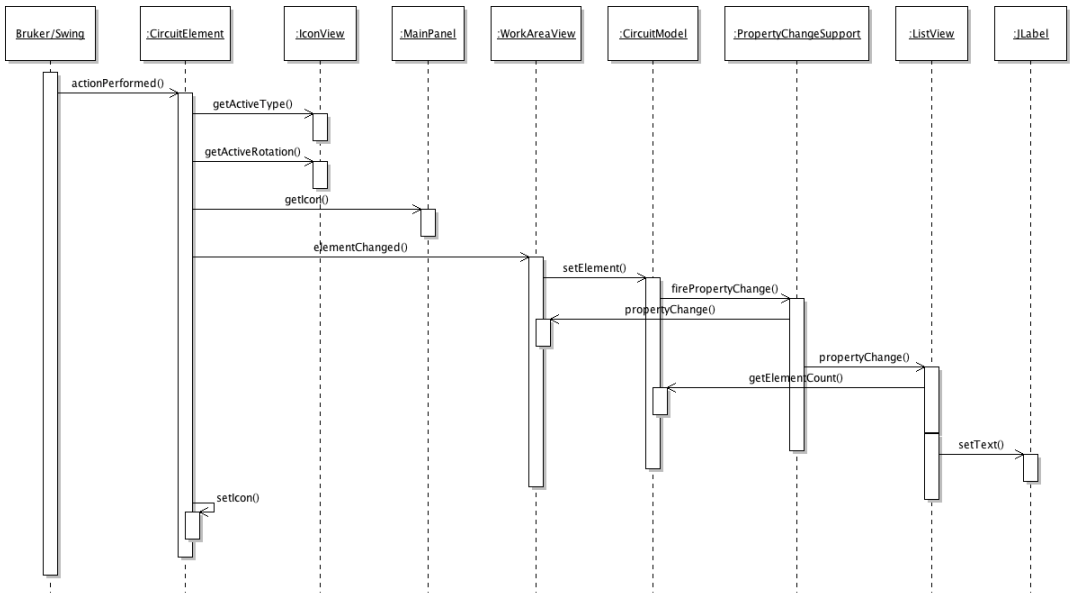
Radio-button-oppførselen mellon ikonknappene kommer automatisk ettersom alle IconButton-objekter er lagt til samme ButtonGroup-objekt.

g) Tegn opp sekvensdiagrammet når brukeren roterer aktivt ikon.



I sekvensidiagrammet over så trykker brukeren på høyre museknapp i et ikon. Metoden mouseClicked() kalles, og IconButton-objektet finner ut hvilken museknapp som er trykket ved å spørre det MouseEvent som ble sendt over som parameter. Dersom det er høye musknapp så øker det verdien av sitt felt "rotation" med en (modulus 4), og henter det riktige Icon grafiske objektet fra sitt MainPanel. Det gir så sitt IconView beskjed om hva som er aktiv rotasjon,og oppdaterer sitt grafiske ikon.

h) Tegn opp sekvensdiagrammet når brukeren plasserer ut en byggesten (valgt ikon) på rutenettet.

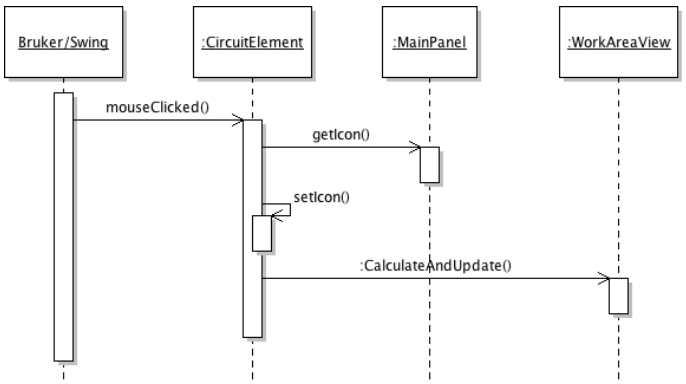


I sekvensdiagrammet over ser vi hva som skjer når brukeren klikker på et CircuitElement (som er en subklasse av JButton):

- *Swing starter metoden actionPerformed i objektet.*
- *Det spør IconView om hva som er aktivt tegneikon, både ikontype og rotasjon.*
- *Det ber så MainPanel om det tilsvarende grafiske ikonet, for å kunne vise riktig ikon. Det gir så beskjed til sitt WorkAreaView om at det er endret, med sin x- og y-posisjon, og ny verdi for type og rotasjon.*

- *WorkAreaView oppdaterer så med riktig verdi tilsvarende i sin CircuitModel.*
- *CircuitModel oppdaterer sine lokale variable, og sender beskjed til sitt PropertyChangeSupport-objekt om at det har skjedd en endring.*
- *PropertyChangeSupport-objektet sender så endringsmeldinger til de som er lyttere, d.v.s ListView og WorkAreaView.*
- *WorkAreaView-objektet mottar endringshendelse, men behøver ikke bruke den til noe. (I en mer kompleks løsning der andre deler av systemet også oppdaterte modellen ville WorkAreaView måtte oppdatere de grafiske ikonene i sine elementer).*
- *ListView får en endringshendelse, og spør modellen om statistikk for hver av de 5 typene.*
- *Alle de 5 JLabel-objektene oppdateres med riktig verdi fra modellen. Her er kun vist første JLabel-objekt.*

i) Tegn opp sekvensdiagrammet når brukeren endrer bryteren i eksempelet fra åpen til lukket slik at lampen begynner å lyse. Du skal ikke beskrive mekanismen for å beregne om det kommer strøm til lampen eller ikke.



Når brukeren klikker med høyre museknapp i et bryterikon så går det en mouseClicked() melding til elementet. Det henter så riktig ikon fra sitt MainPanel. (Det må finnes en egen byggestentype som er "Lukket bryter"). Elementet gir så beskjed til sit WorkAreaView om å beregne og oppdatere i forhold til hvilke lamper som skal lyse utifra om det går strøm eller ikke. På samme måte så må det også finnes en egen byggestentype for lampe som lyser).

Karakter hele oppgave 3:

A. *Løsning der modellen er gjort korrekt, og der det er forklart hvordan aktivt tegneikon er handtert..*
B. *Modellen er gjort korrekt.*
C. *Modellen er satt opp riktig, men men noen feil.*
D. *Store mangler*
E. *Veldig store mangler.*
F. *Ikke besvart, eller ikke relevant svar.*

Du trenger ikke forholde deg til layout eller hvordan komponentene tegnes ut.

For hver av komponentklassene så er relevante metoder og hjelpeklasser listet i appendikset bakerst. Merk: Det forlanges ikke at du skal anvende metoder eller klasser som ikke er listet i appendikset.

Hint: Et JPanel eller en subklasse av JPanel kan godt være usynlig (ha samme farge og rammefarge som bakgrunnen) og brukes til å samle flere Swing-komponenter.

Relevante klasser, interface og tilhørende metoder

Det følgende er klippet og limt fra den offisielle dokumentasjonen på java.sun.com.

class PropertyChangeSupport

This is a utility class that can be used by objects that support bound properties. You can use an instance of this class as a variable in your object and delegate various work to it.

Methods:

public void addPropertyChangeListener(PropertyChangeListener listener)

Add a PropertyChangeListener to the listener list. The listener is registered for all properties.
Parameters:
listener - The PropertyChangeListener to be added

public void removePropertyChangeListener(PropertyChangeListener listener)

Remove a PropertyChangeListener from the listener list.
Parameters:
listener - The PropertyChangeListener to be removed

public void firePropertyChange(String propertyName,
Object oldValue,
Object newValue)

Report a bound property update to any registered listeners. No event is fired if old and new are equal and non-null.
Parameters:
propertyName - The name of the property that was changed.
oldValue - The old value of the property.
newValue - The new value of the property.

interface PropertyChangeListener

A "PropertyChange" event gets fired whenever an object changes a "bound" property. You can register a PropertyChangeListener with a source object so as to be notified of any bound property updates.

Methods:

public void propertyChange(PropertyChangeEvent evt)

This method gets called when a bound property is changed.
Parameters:
evt - A PropertyChangeEvent object describing the event source and the property that has changed.

class PropertyChangeEvent

A "PropertyChange" event gets delivered whenever an object changes a "bound" or "constrained" property. A PropertyChangeEvent object is sent as an argument to the PropertyChangeListener method. Normally PropertyChangeEvents are accompanied by the name and the old and new value of the changed property. Null values may be provided for the old and the new values if their true values are not known. An event source may send a null object as the name to indicate that an arbitrary set of if its properties have changed. In this case the old and new values should also be null.

Methods:

public String getPropertyName()

Gets the programmatic name of the property that was changed.
Returns:
The name of the property that was changed. May be null.

public Object getNewValue()

Gets the new value for the property, expressed as an Object.
Returns:
The new value for the property. May be null

public Object getOldValue()

Gets the old value for the property, expressed as an Object.
Returns:
The old value for the property. May be null.

class JPanel

JPanel is a generic lightweight container.

Methods:

public void addMouseListener(MouseListener l)

Adds the specified mouse listener to receive mouse events from this component.
Parameters:
l - the mouse listener

interface MouseListener

The listener interface for receiving "interesting" mouse events (press, release, click, enter, and exit) on a component.
The class that is interested in processing a mouse event implements this interfac).
The listener object created from that class is then registered with a component using the component's addMouseListener method. A mouse event is generated when the mouse is pressed, released clicked (pressed and released). When a mouse event occurs, the relevant method in the listener object is invoked, and the MouseEvent is passed to it.

Methods:

public void mouseClicked(MouseEvent e)

Invoked when the mouse button has been clicked (pressed and released) on a component.

class MouseEvent

An event which indicates that a mouse action occurred in a component.

Methods:

public Object getSource()

The object on which the Event initially occurred.
Returns:
The object on which the Event initially occurred.

public int getButton()

Returns which, if any, of the mouse buttons has changed state.
Returns:
one of the following constants: NOBUTTON, BUTTON1, BUTTON2 or BUTTON3.

class JButton

An implementation of a push button

Methods:

public JButton(Icon icon)
Creates a button with an icon.
Parameters:
icon - the image that the button should display

public void addActionListener(ActionListener l)

Adds the specified action listener to receive action events from this JButton.

public void setIcon(Icon defaultIcon)

Sets the button's icon.

class JToggleButton

An implementation of a two-state button -- an item that can be selected or deselected, and which displays its state to the user. Used with a ButtonGroup object to create a group of buttons in which only one button at a time can be selected. (Create a ButtonGroup object and use its add method to include the JToggleButton objects in the group.)

Methods:

public JToggleButton(Icon icon)
Creates an initially unselected toggle button with the specified image but no text.
Parameters:
icon - the image that the button should display

public void addActionListener(ActionListener l)
Adds the specified action listener to receive action events from this JToggleButton.

public boolean isSelected()
Returns the state of the button. True if the toggle button is selected, false if it's not.
Returns:
true if the radiobutton is selected, otherwise false

public void setSelected(boolean b)
Sets the state of the radiobutton. Note that this method does not trigger an actionEvent.
Parameters:
b - true if the button is selected, otherwise false

public void setIcon(Icon defaultIcon)
Sets the button's icon.

interface ActionListener

The listener interface for receiving action events. The class that is interested in processing an action event implements this interface, and the object created with that class is registered with a component, using the component's addActionListener method. When the action event occurs, that object's actionPerformed method is invoked.

Methods:

public void actionPerformed(ActionEvent e)
Invoked when an action occurs.

Class ActionEvent

A semantic event which indicates that a component-defined action occured. This high-level event is generated by a component (such as a Button) when the component-specific action occurs (such as being pressed). The event is passed to every ActionListener object that registered to receive such events using the component's addActionListener method.
The object that implements the ActionListener interface gets this ActionEvent when the event occurs. The listener is therefore spared the details of processing individual mouse movements and mouse clicks, and can instead process a "meaningful" (semantic) event like "button pressed".

Methods:

public Object getSource()

The object on which the Event initially occurred.
Returns:
The object on which the Event initially occurred.

class ButtonGroup

This class is used to create a multiple-exclusion scope for a set of buttons. Creating a set of buttons with the same ButtonGroup object means that turning "on" one of those buttons turns off all other buttons in the group. A ButtonGroup can be used with any set of objects that inherit from AbstractButton. Typically a button group contains instances of JRadioButton, JRadioButtonMenuItem, or JToggleButton. Initially, all buttons in the group are unselected.

Methods:

public void add(AbstractButton b) *(for example JRadioButton)*

Adds the button to the group.
Parameters:
b - the button to be added

class JLabel

A display area for a short text string. A label does not react to input events.

Methods:

public void setText(String text)

Defines the single line of text this component will display.