Heisrapport gruppe 76

Tobias Lavik og Øyvind Ystanes Dolmen

Innhold:

1. Moduler og virkemåte

Programmet består av 5 moduler/.h-filer:

- Heispanel.h
- Kø.h
- Heiskontroller.h
- Heistilstand.h
- Dør.h

Heispanel modulen har i oppgave å lytte etter input i heispanelet og etasjepanelet fra bruker og kalles kontinuerlig i main-funksjonen.

Kø modulen håndterer etasjer som legges til i køen og sletting av etasjer. Køen er en dynamisk allokert liste som vokser og krymper sammen med lengden på køen. Køen kan derfor skaleres opp sammen med antall etasjer dersom den skulle blitt brukt i en annen heismodell med flere etasjer. Etasje er en struct som inneholder etasjenummer og retning. Kø er en struct som inneholder lengden på køen og en peker til stedet i minne der etasjer av typen Etasje lagres. Funksjonene som endrer på køen må derfor allokere og frigjøre minne.

```
typedef struct {
  int etasje;
  int retning;
} Etasje;

typedef struct{
    Etasje* liste;
    int lengde;
} Kø;
```

Heiskontroller modulen har i oppgave å kontrollere den fysiske heismodellen. Den har derfor i oppgave å kalle elevio-funksjonene for lys og kjøring av heis.

Heistilstand modulen holder variabler for hvor heisen er, om den er i bevegelse og retning. Modulen sjekker også om heisen er i etasjen sendt som parameter i funksjonskallet.

Dør modulen har ansvar for nedtellingsfunksjonen som brukes når døren går opp og under trykk på stoppknappen. Funksjonen start_nedtelling() starter en 3 sekunders while løkke som sjekker for eventuelle inputs fra bruker slik som obstruksjoner, stoppknapp eller etasjer som skal legges til i køen.

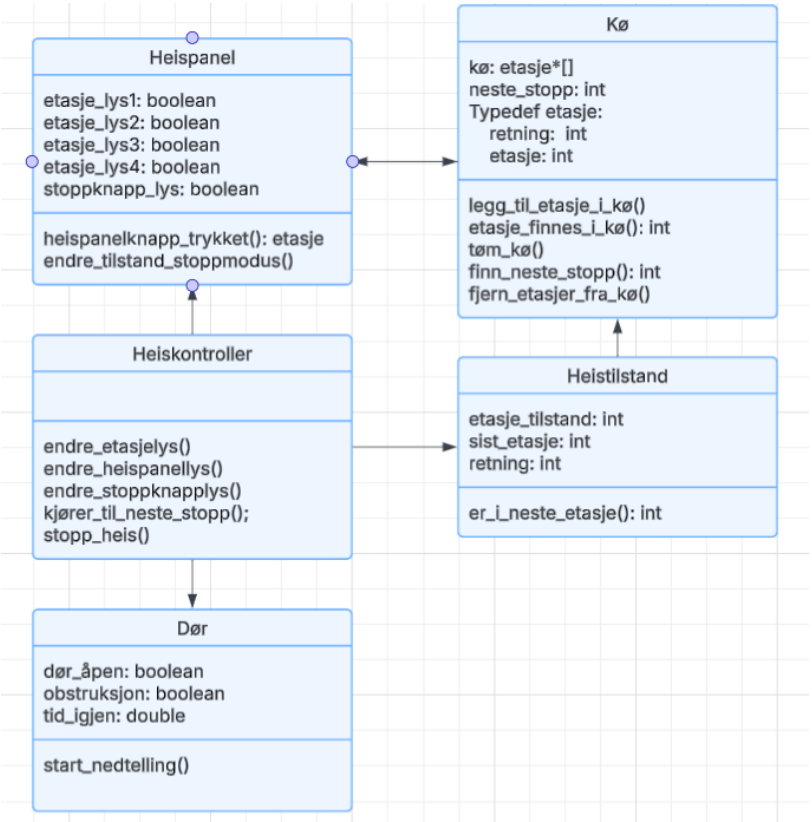
Main funksjonen består av en evig while-løkke som kjører så lenge stoppknappen ikke er aktiv. Når programmet starter, initialiseres elevio og en tom kø. Heisen kjører også til default-posisjon. I while-løkka oppdateres kontinuerlig variablene etasje_tilstand og neste_stopp slik

at heisen kan kjøre mot det neste og riktige stoppet gitt kravene til systemet. While-løkka setter også lys og sjekker om heisen har kommet til neste_stopp kontinuerlig. Dersom stoppknappen trykkes bryter koden ut av while-løkka og gjør kravene gitt til håndtering av trykk på stoppknappen. Når stoppknappen ikke lenger er aktiv kommer koden tilbake i while-løkka og fortsetter slik som beskrevet over.

2. Refleksjoner rundt bruk av V-modellen og UML

Bruken av den pragmatiske V-modellen og UML-diagrammer bidro til en mer strukturert utviklingsprosess, der vi allerede i startfasen ble tvunget til å tenke gjennom systemarkitektur. Dette ga oss et helhetlig overblikk over systemets oppbygning og funksjonalitet.

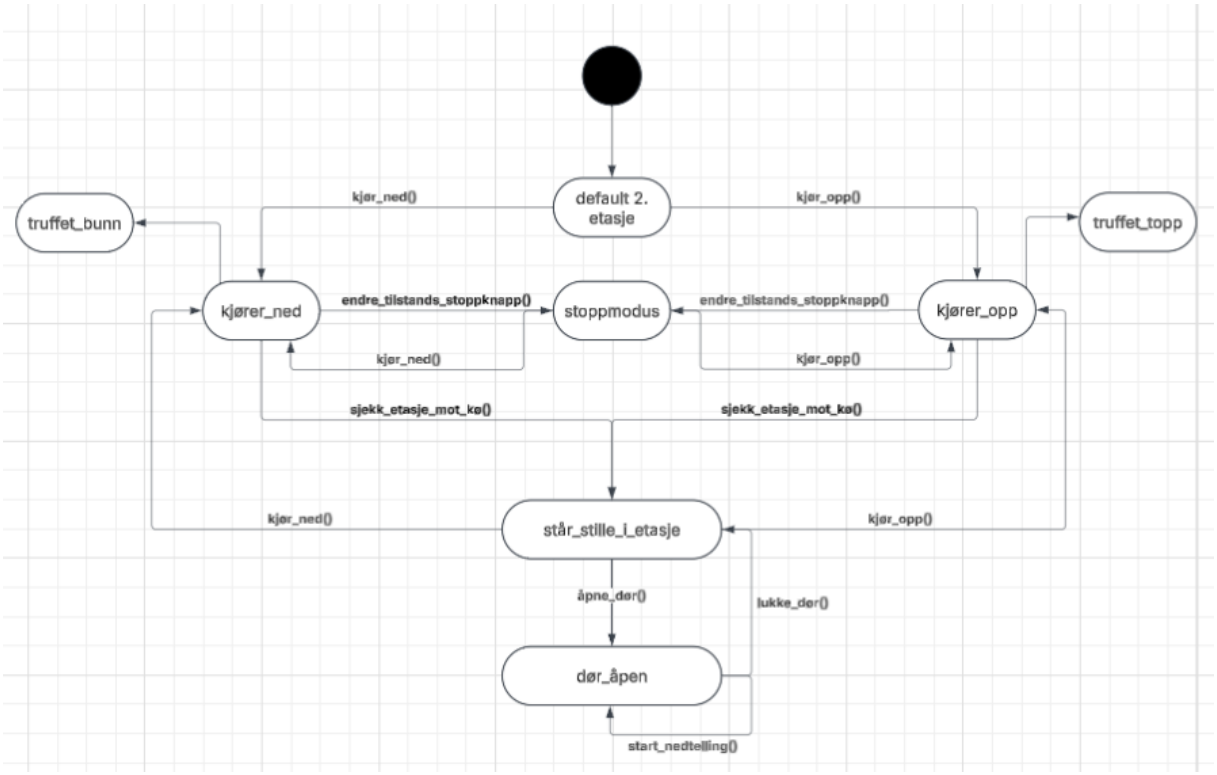
Figur 1 viser et klassediagram brukt som mal til koden. Det er noen variasjoner mellom ferdig kode og denne malen, for eksempel i funksjonsnavn.



Figur 1: klassediagram

Selv om denne prosessen la et solid grunnlag, opplevde vi at en del av de opprinnelige planene måtte justeres underveis. Etter hvert som vi begynte å implementere koden, avdekket vi alternative løsninger som gjorde enkelte av de tidlige valgene mindre relevante. Likevel var arbeidet med V-modellen og UML verdifullt, spesielt i forhold til systemforståelse.

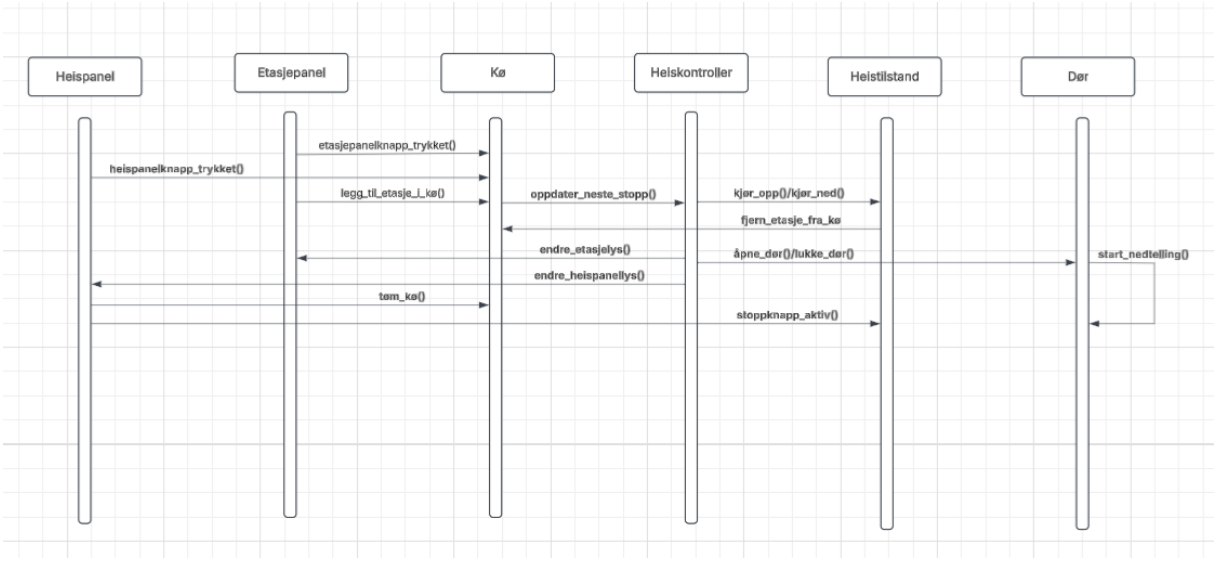
Figur 2 viser et tilstandsdiagram av heisen. Merk at noen av funksjonene slik som kjør_opp() og kjør_ned() har blitt slått sammen til kjør_til_neste_stopp(). Logikken i tilstandsdiagrammet står seg likevel.



Figur 2: tilstandsdiagram

Vi erfarte at klassediagrammet og tilstandsdiagrammet var de mest nyttige verktøyene. Klassediagrammet hjalp oss med å organisere kodens struktur, mens tilstandsdiagrammet gjorde det enklere å forstå systemets tilstandsendringer. Derimot viste sekvensdiagrammet seg å være noe mindre nyttig, da det fokuserer på mer detaljert interaksjon mellom ulike moduler i programmet, noe som endret seg underveis i implementeringen.

Figur 3 viser sekvensdiagrammet brukt som mal for koden. Merk at heispanel og etasjepanel modulene ble slått sammen til en modul, heispanel.



Figur 3: sekvensdiagram

Videre ga den pragmatiske V-modellen oss en viktig tilnærming til modularisering, noe som bidro til både skalerbarhet og enklere vedlikehold. Ved å tidlig identifisere og strukturere systemets moduler kunne vi sikre en ryddig kodebase, noe som forenklet senere justeringer og videreutvikling.

3. Bruk av kunstig intelligens

Vi benyttet kunstig intelligens (KI) i begrenset grad, hovedsakelig til feilsøking og spørsmål knyttet til programmering i C. KI ble blant annet brukt til å identifisere feil i koden og forklare hvordan ulike konstruksjoner som structs og pekere fungerer.

Den mest betydningsfulle påvirkningen av KI i prosjektet var i implementeringen av køsystemet. Her foreslo KI å bruke en pekerbasert datastruktur, noe vi syntes virket fornuftig. I etterkant innså vi at en matrisebasert løsning kunne vært mer hensiktsmessig, spesielt med tanke på kodeforståelse og enkelhet, gitt at systemet kun håndterer fire etasjer. Dette ville også fjernet risikoen med minneallokering. På den andre siden skal det også nevnes at en dynamisk køstruktur, slik den vi endte opp med, har fordelen av skalerbarhet. Dersom

systemet skulle utvides til for eksempel 20 etasjer, ville en pekerbasert løsning vært mer fleksibel enn en statisk matrise.

Samlet sett erfarte vi at KI kunne være et nyttig verktøy for spesifikke oppgaver, men at kritisk vurdering av forslagene er nødvendig for å sikre optimale løsninger i praksis.