Universitetet i Stavanger

DET TEKNISK-NATURVITENSKAPELIGE FAKULTET

BACHELOROPPGAVE

Studieprogram/spesialisering: Datateknologi	Vårsemesteret, 2021 Åpen
Forfattere: Brage Riis Gundersen Fredrik Netteland	Signaturer 
Fagansvarlig og veileder: Erlend Tøssebro	
Tittel: System for elektronisk innlevering og retting av tre- og grafoppgaver i DAT200	
Studiepoeng: 20	
Emneord: Nettbasert Tegneverktøy Automatisk Retting Trær og grafer	Sidetall: 57 + Vedlegg: Zip-arkivfil Stavanger, fredag 14. mai 2021

Innhold

Del I

Sammendrag

Oppgaven går ut på å lage et nettbasert system for automatisk retting av DAT200-oppgaver. Applikasjonen vi har laget gir faglærer mulighet til å enkelt lage oppgavesett som elever kan løse, samtidig som elever får en bedre forståelse av emnet. For å løse oppgavene bygger og manipulerer studentene ulike datastrukturer i et grafisk grensesnitt. Oppgavene rettes ved å sammenligne studentens svar mot en fasit, og en oversikt over innleveringer er tilgjengelig for administrator. Resultatet er et helhetlig produkt som tar hånd om hele oppgaveprosessen fra oppgavelaging til innlevering og oversikt.

Strukturbyggeren er tilgjengelig på: `strukturbyggeren.ux.uis.no`

Del II

Innledning

1 Oppgaven

Oppgaven gikk ut på å lage en applikasjon knyttet til øvingsoppgavene i faget DAT200-Alogritmer og datastrukturer. Applikasjonen skal presentere oppgaver som omhandler trær, hauger og grafer for studentene og skal kunne løses av studenter ved å tegne trær eller grafer. Svarene skal kunne rettes ved å sammenligne mot et løsningsforslag. Det skal også være mulig å bruke applikasjonen til å øve og lære i emnet, gjennom å se på eksempler og ved å løse oppgaver.

2 Målsetting

- Lag et system som lar studentene tegne og modifisere trær og grafer på skjerm.
- Sammenlikne to grafer og tegne ut forskjellen på skjerm.
- Sammenlikne oppgaver som har flere trinn.
- Visualisere resultat av retting.
- Implementere Feide-innlogging
- Integrere med Canvas.

3 Motivasjon

Motivasjonen for oppgaven ligger i å gjøre retting av tre- og grafoppgaver enklere. I tillegg er noe av motivasjonen å øke forståelsen og intuisjonen rundt disse strukturene ved å tilby gode oppgaver og et godt verktøy til elevene. Ved å lage et robust system kan vi gjøre det mulig for faglærer (eller administrator) å enkelt lage oppgaver rundt ønsket struktur. Deretter skal elever kunne laste inn oppgavene og rette etter fasit, mens faglærer enkelt kan få oversikt over innleveringene.

4 Arbeidsflyt

For å samarbeide godt er det viktig å velge en god digital løsning slik at man raskt kan dele filer, endringer og kode. Vi valgte å bruke det populære versjonkontrollprogrammet Git sammen med tjenesten GitHub, særlig fordi vi har erfaring med programmet fra før. I tillegg har Git funksjonalitet for å lage grener samt mange andre funksjoner som gjør det enklere å utvikle sammen. Dokumentet skriver vi i LaTeX-formatet med skriveprogrammet LyX etter anbefaling fra instituttleder. Jobbingen på prosjektet foregikk i alle ukedager utenom torsdager hvor vi snakket vi sammen over TeamSpeak3.

Del III

Bakgrunn

5 HTML, JavaScript og CSS

Hvilket rammeverk man bruker har mye å si for arbeidsflyten, derfor prioriterte vi et rammeverk som passer på de aller fleste plattformer, nemlig HTML, JavaScript og CSS. Selv om applikasjonen passer best for mus og tastatur, og gjerne kunne vært en klassisk kompilert skrivebordapplikasjon for Windows eller Mac, krever dette ekstra arbeid siden det er få rammeverk som dekker begge plattformene. For eksempel kunne man laget applikasjonen med rammeverk som WPF eller UWP for Windows eller Cocoa for macOS, men her hadde kanskje Qt vært et bedre alternativ, ettersom det er tilpasset begge plattformer. En annen fordel med nettapplikasjoner er at klienten alltid vil få den nyeste koden fra tjeneren slik at det er lettere å gi ut programvareoppdateringer. Når det gjelder nettapplikasjoner, finnes det mange store og populære rammeverk som React.js, Angular og ASP.NET, og nykommere som Blazor. Her har vi sett en utvikling vekk fra de tradisjonelle «flersideapplikasjonene» bestående av flere statiske sider levert av tjeneren, mot «enkeltsideapplikasjoner». Disse ligner mer på ordinære kompilerte programmer i det at det ikke er flere statiske sider som lastes inn og som det navigeres i mellom, men heller én side som manipuleres. Vi har prioritert rammeverk for den største plattformen som finnes, nemlig nettleseren, og valgt å holde det enkelt med de mest standardiserte rammeverkene. Derfor kan vi være sikre på at vi kan bruke applikasjonen i lang tid fremover.

5.1 Canvas-elementet

For å visualisere trær og grafer, bruker vi i denne applikasjonen et «HTML Canvas»-element. Elementet er et område med piksler som kan manipuleres, men i utgangspunktet finnes det bare primitive måter å tegne på, strek eller bue mellom to koordinater, eller tekst. Utenom det er all funksjonalitet implementert av oss. Det finnes også store rammeverk som utvider funksjonaliteten, men vi har valgt å skreddersy dette selv, dermed er vi ikke avhengige av rammeverk som vi ikke har kontroll over.

En av «Canvas»-elementets nyttige funksjoner er at man kan høyreklikke og velge «Save as» for å lagre et bilde av det som er tegnet.

5.2 JavaScript og JSON

JavaScript har den fordelen at språket er inkludert i nettlesere, slik at brukere ikke trenger å laste ned ny programvare for å bruke applikasjonen. Dessuten er det lett å jobbe med JSON-formatet når man bruker JavaScript.

JSON står for «JavaScript Object Notation» og brukes til lagring av data i form av objekter, formatet er også strukturert for å kunne være leselig for mennesker. JSON er ideelt når mindre datamengder skal lagres, slik tilfellet er for denne applikasjonen. Applikasjonen skal kunne gi brukeren oppgaver og rette disse (ecma international, 2017).

6 Feide

Ettersom applikasjonen skal la administratorer sjekke hvilke oppgaver studentene har løst, trengte vi en måte å autentisere brukere på. Et innloggingssystem er noe vi kunne prøvd i implementere selv, men da ville vi trengt mye tid på å gjøre systemet sikkert, derfor valgte vi å bruke Feide. Feide er en innloggingstjeneste levert av Uninett, og brukes i mange applikasjoner relatert til utdanning i Norge. Feide er et «Single Sign On»-system som betyr at en bruker kan logge på en Feide-applikasjon, og da slippe å logge inn på en annen applikasjon som også bruker Feide (Uninett, u.å.).

7 Python og Flask

Tjenerkoden er skrevet i Python. Grunnen til at vi valgte Python fremfor andre løsninger, som Node.js, er at vi har erfaring med Python og Flask fra før av, og at Flask er et veldig «lett» rammeverk som ikke inkluderer mye vi ikke har behov for. Samtidig har Flask støtte for utvidelser vi faktisk trenger, som database- og innloggingsmoduler.

7.1 Authlib

Authlib er et Python-bibliotek som gjør det mulig for Python-baserte webservere som Flask og Django å bruke OAuth og OpenID Connect-baserte tjenester. (lepture, 2021).

7.1.1 OAuth 2.0

OAuth 2.0 er en protokoll som brukes til å autentisere brukere via en autentiseringstjener. OAuth 2.0 lar flere OAuth-klienter (applikasjoner) bruke en felles innloggingstjeneste for å håndtere autentiseringen. Dette gjør at OAuth-klienter ikke trenger å implementere et eget autentiseringssytem som kan inneholde svakheter. OAuth-klienten lagrer heller ikke brukerne, det vil si at den holder på mindre potensielt sensitiv informasjon som den ellers måtte ha lagret på en trygg måte. OAuth fungerer ved at ressurseieren (brukeren) blir omdirigert til en tjener som håndterer autentisering. Sammen sendes blant annet informasjon om hvilken data OAuth-klienten vil ha fra ressurseieren kalt «scope», og hvilken side ressurseieren skal bli omdirigert til etter innlogging. Her blir ressurseieren vist til innloggingstjenestens innloggingsside, og må skrive inn brukernavn og passord. Ressurseieren må også godta at OAuth-klienten skal få tilgang til etterspurte data. Ressurseieren blir etterpå omdirigert til den vedlagte siden sammen med en midlertidig kode fra autentiseringstjeneren. Denne koden kan OAuth-klienten sende til autentiseringstjeneren for å få tilgang til den forespurte dataen. Diverse koder som identifiserer og gir OAuth-klienten tilgang, må også sendes med. Gitt at autentiseringstjeneren godkjenner de tilsendte kodene, blir OAuth-klienten tilsendt et adgangskort. Kortet beviser at OAuth-klienten har tilgang til den forespurte brukerdataen om ressurseieren. OAuth-klienten vedlegger denne koden når den henter data fra en ressursserver hvor brukerdata ligger. (OktaDev, 2019, 8:10).

7.1.2 OpenID Connect

OpenID Connect bygger på OAuth 2.0 og lar OAuth-klienter hente data om den autentiserte ressurseieren. For at en autentiseringstjener skal vite at OpenID Connect skal brukes, sendes «openid» med som et «scope». Dette gjør at OAuth-klienten senere vil bli tilsendt et identitetsbevis, som inneholder data om ressurseieren. (OktaDev, 2019, 10:57).

7.2 Flask-login

Når vi bestemte oss for å bruke Feide-tjenesten til innlogging, trengte vi en måte å bruke dataene gitt fra en Feide-innlogging til å gi den autentiserte brukeren tilgang til applikasjonen, mens ikke-autentiserte brukere ikke skulle ha tilgang. For å gjøre dette brukte vi Flask-login, som er et Python-bibliotek som kan brukes til å logge brukere inn eller ut ved hjelp av «sessions». Flask-login kan sørge for at brukere ikke kan besøke applikasjonens sider dersom de ikke er logget inn. Flask-login er en utvidelse av Flask og kan brukes til å logge brukere inn og ut ved hjelp av et økt-system («sessions») som også sørger for at brukere som ikke er innlogget, blir avvist (Countryman, 2019).

7.3 Jinja

Jinja (davidism. ThiefMaster, 2021a) er et språk som kan brukes til å skrive «templates» for blant annet Flask-tjeneren. En «template» er en HTML-fil som i tillegg til vanlig HTML-kode inneholder Jinja-kode. Når en «template» lastes inn, kan en webtjener sende med data. Ved å bruke Jinja-kode, kan denne dataen dynamisk legges til i HTML-dokumentet. Jinja-kode kan også inneholde sjekker og løkker. Jinja vil også sørge for at data som vises dynamisk, ikke vil bli tolket som kode. Dermed er det trygt å vise data innsendt av brukere ved hjelp av Jinja. (davidism. ThiefMaster, 2021b).

Del IV

Konstruksjon

8 Startside

Ved å trykke på en lenke til Strukturbyggeren havner du på startsiden hvor du kan gå videre til innloggingen.

Figur 1: Startsiden.

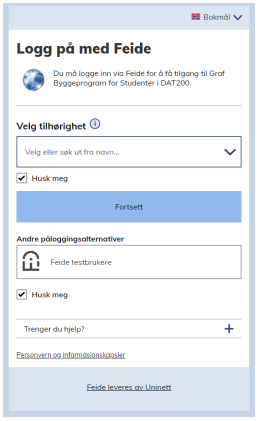
9 Innloggingsside

Feide-innlogging blir brukt for autentisere brukere av applikasjonen. Ved første innlogging blir brukeren registrert i databasen. Etter at brukeren er blitt autentisert gjennom Feide-tjenesten brukes Flask-login-modulen til å holde styr på innloggede brukere. Hvorvidt brukeren har tilgang til administratorfunksjonalitet bestemmes av om brukeren er registrert som administrator i databasen, og dette vil bli sjekket ved innlogging.

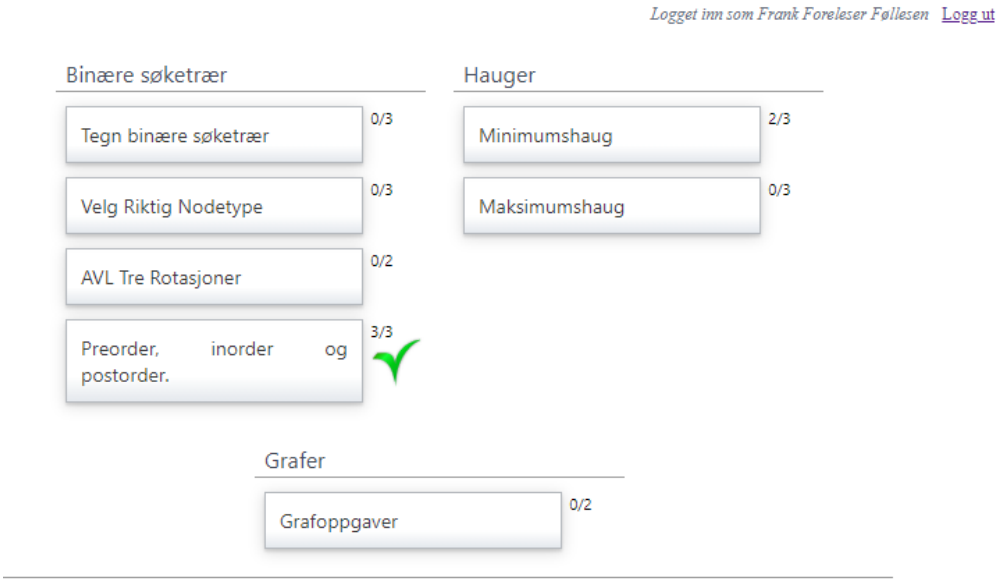
10 Oppgaveside

Oppgavesiden er delt inn i seksjoner der hver seksjon er et oppgavesett. Innenfor oppgavesettet ligger oppgavene. Ved siden av oppgavene kan vi se hvor mange deloppgaver som er blitt laget for oppgaven, og hvor mange brukeren har løst. Har man løst alle deloppgavene, får man en grønn hake ved siden av oppgaven.

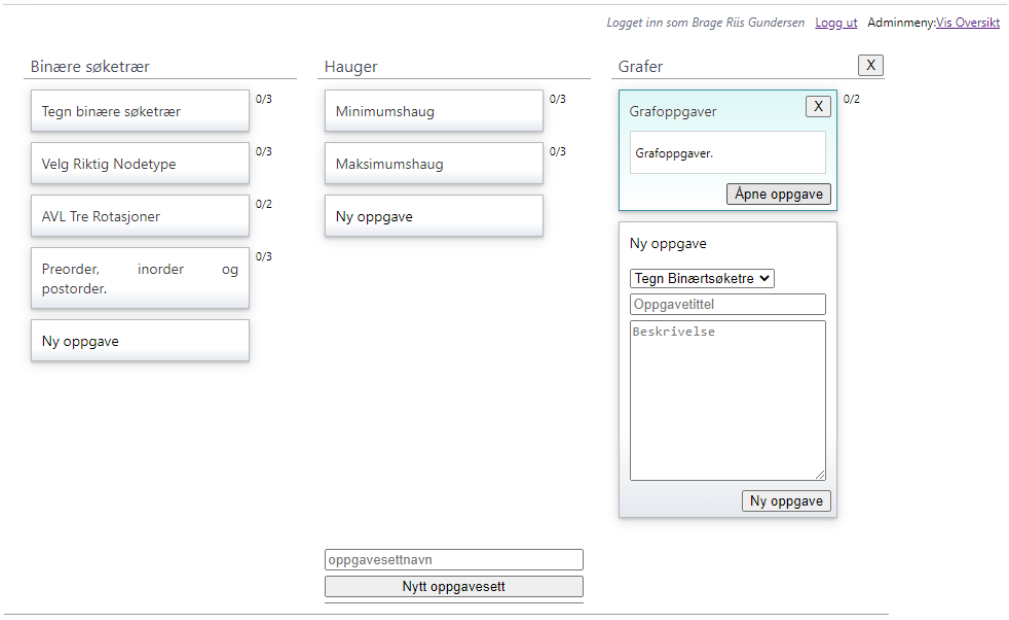
Om man er logget inn som en administrator, ser oppgavesiden noe anderledes ut. Som administrator har man mulighet å legge til eller fjerne oppgavesett og oppgaver direkte gjennom brukergrensesnittet. For å legge til deloppgaver må man åpne en av oppgavene.



Figur 2: Logginnside levert av Uninett.



Figur 3: Oppgavesiden logget inn som bruker. Brukeren har trykket på ≪Grafoppgaver≫-boksen.



Figur 4: Oppgavesiden logget inn som administrator. Administratoren har trykket på ≪Grafoppgaver≫-boksen og ≪Ny oppgave≫-boksen.

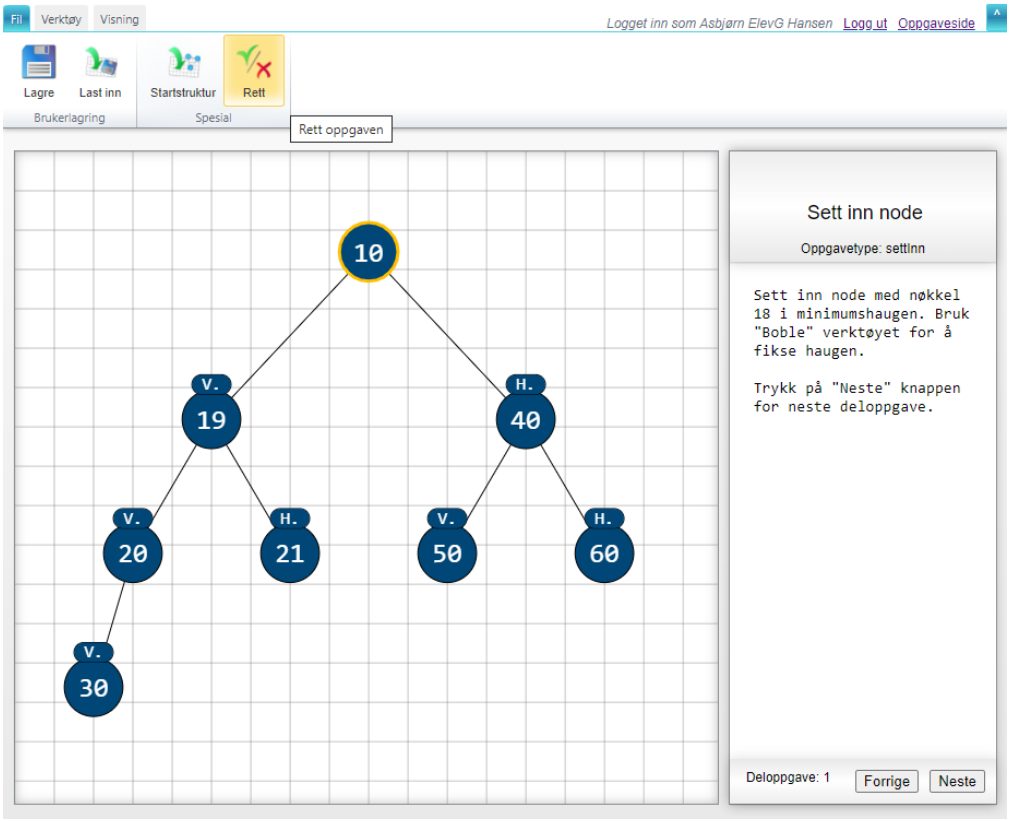
11 Oversiktsside

Som administrator kan man åpne oversiktsiden for å få en oversikt om hvilke brukere som har levert inn hvilke deloppgaver. Man kan sortere tabellen ved å trykke på kolonneoverskriften. I tillegg kan man søke gjennom tabellen, selve søket skjer da bare på klientsiden, og rader som ikke inneholder det man søker på, blir luket ut. Radene kommer tilbake når man fjerner tekst fra søkefeltet igjen. På den måten fungerer søket som et filter, og man sparer databasen for mye arbeid da søket skjer på klientsiden i tillegg til at klienten ikke må vente på respons fra tjeneren.

Figur 5: Oversiktssiden.

12 Strukturbyggeren

Strukturbyggeren er delen av applikasjonen som brukes til å visualisere og bygge trær, hauger og grafstrukturer. Strukturbyggeren inkluderer verktøylinje og et lerret. Strukturbyggeren er grunnstenen i applikasjonen, og det er den som skal gjøre det lett å tegne opp ulike strukturer. At verktøyene er intuitive å bruke er svært viktig for brukeropplevelsen, brukermanual skal ikke være nødvendig for å ta i bruk applikasjonen. Av hensyn til opplevelsen er det derfor implementert ulike animasjoner som forteller om hva som skjer i lerretet og skal gjøre det lett å forstå hvordan verktøyene skal brukes.



Figur 6: Strukturbyggeren, musepeker er over ≪Rett≫-knappen.

12.1 Lerretet

12.1.1 Noder

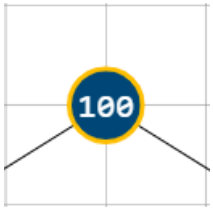
En node er definert som en klasse som inneholder forskjellige attributter. Ettersom en node kan brukes til å visualisere flere datastrukturer, vil enkelte attributter bare brukes i noen sammenhenger. To attributter som alltid brukes, er nodens x og y-verdier, som brukes til plassering på lerretet. Når brukeren plasserer en ny node på lerretet, vil den nye noden få x og y-verdien til musepekeren, og noden lagres i en liste av noder.

Figur 7: Enkel node.

Figur 8: Rettet kant mellom vektede noder.

12.1.2 Noder i binære søketrær

Noder kan blant annet brukes til å visualisere nodene i et binært søketre, og en av disse nodene kalles rotnoden. Rotnoden er den øverste noden i treet, og visualiseres med en gul ring (se figur 9).



Figur 9: Rotnoden i et binært søketre visualisert med gul ring.

I denne applikasjonen representerer en node en nøkkel, og visualiseres med tallet i midten av noden. Tallet kan bli satt av brukeren, og i binære søketrær er bare heltall tillatt. Dersom brukeren ønsker å sette negative verdier, kan «-» tasten brukes til å gjøre det nåværende tallet negativt. For å gjøre negative tall til et positivt tall, brukes «+» tasten.

12.1.3 Noder i grafoppgaver

I grafoppgaver brukes også noder for å visualisere strukturen, men teksten representerer ikke nøkler, i stedet vil nodene få en bokstav i alfabetisk rekkefølge i den rekkefølgen de ble plassert. Dette gjøres ettersom rekkefølgen av enkelte grafalgoritmer påvirkes av hvilke noder som ble plassert først.

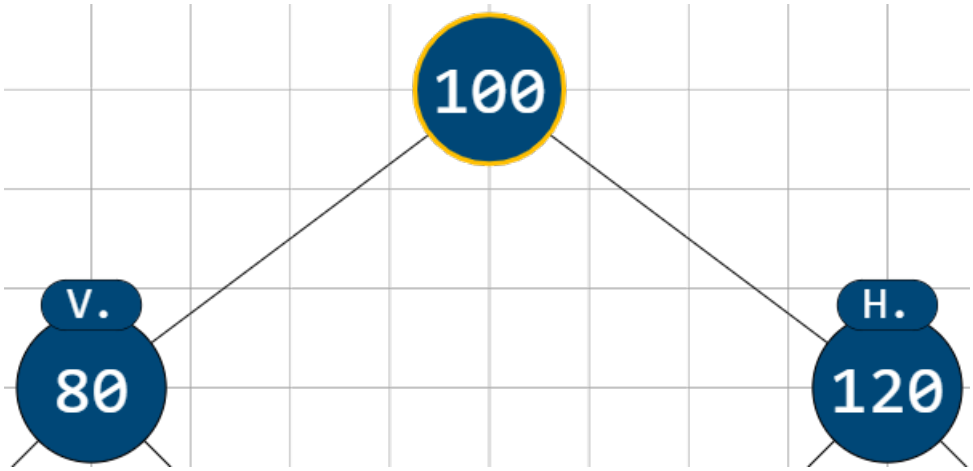
12.1.4 Kanter og grener

En node kan kobles sammen med andre noder med en kant eller gren som vises som en svart strek mellom nodene. En kant er en sammenkobling av to noder og representerer hvordan dataobjekter henger sammen i en datastruktur. En kant er også definert som en klasse som har forskjellige attributter. En kant vil alltid være tilknyttet to noder. Referanser til nodene lagres i kantobjektet og brukes til tegning av kanten. Nye kanter blir lagret i en liste.

Når vi snakker om trestrukturer blir disse kantene kalt grener. Grener lages med den samme klassen som kanter og brukes på samme måte, bortsett fra at grener ikke kan være rettede.

12.1.5 Bønner

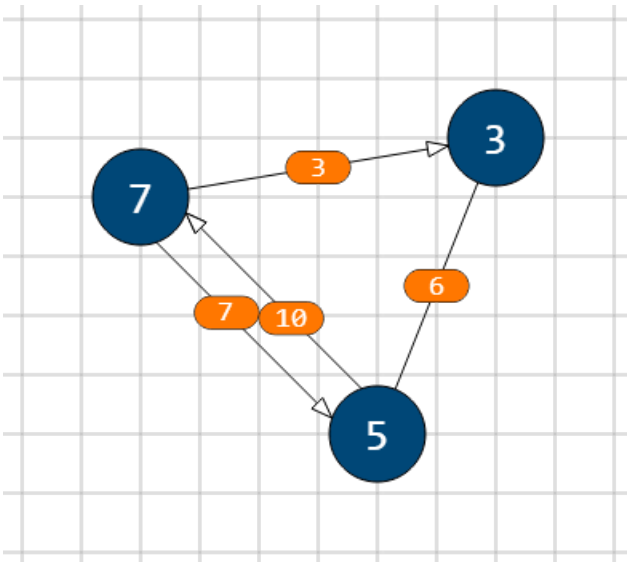
I denne applikasjonen visualiseres diverse informasjon ved hjelp av «bønner». En bønne er i denne applikasjonen et rektangel med avrundede sider som inneholder diverse informasjon. I oppgaver om binære søketrær viser bønnen om en node er et venstre- eller høyrebarn (Se figur 10), mens i grafoppgaver brukes bønner til å vise kantvekter (Se figur 11).



Figur 10: Bønner tegnet ovenfor barnnoder for å visualisere om noden er et høyre- eller venstrebarn.

12.1.6 Vekt

Kantvektene er tegnet oppå kantene som «bønner». Bønnene tegnes bare opp når kanten er vektet, og tegnes som regel midt på kanten. Bønner på dobbelkanter tegnes henholdsvis på 40 og 60% av kantlengden slik at de ikke dekker over hverandre.



Figur 11: Struktur med vektede noder og kanter samt rettede og urettede kanter. I bakgrunnen ser vi rutenettet.

12.1.7 Tegne på nytt

For hver gang det skjer en endring som skal visualiseres på lerretet, må innholdet på lerretet tegnes på nytt. Først tømmes lerretet, og bakgrunnen tegnes. Rutenettet tegnes på toppen, og deretter brukes kantlisten til å tegne alle kantene. Etterpå tegnes nodene slik at de havner over kantene. Til slutt tegnes «bønnene» og andre menyelementer.

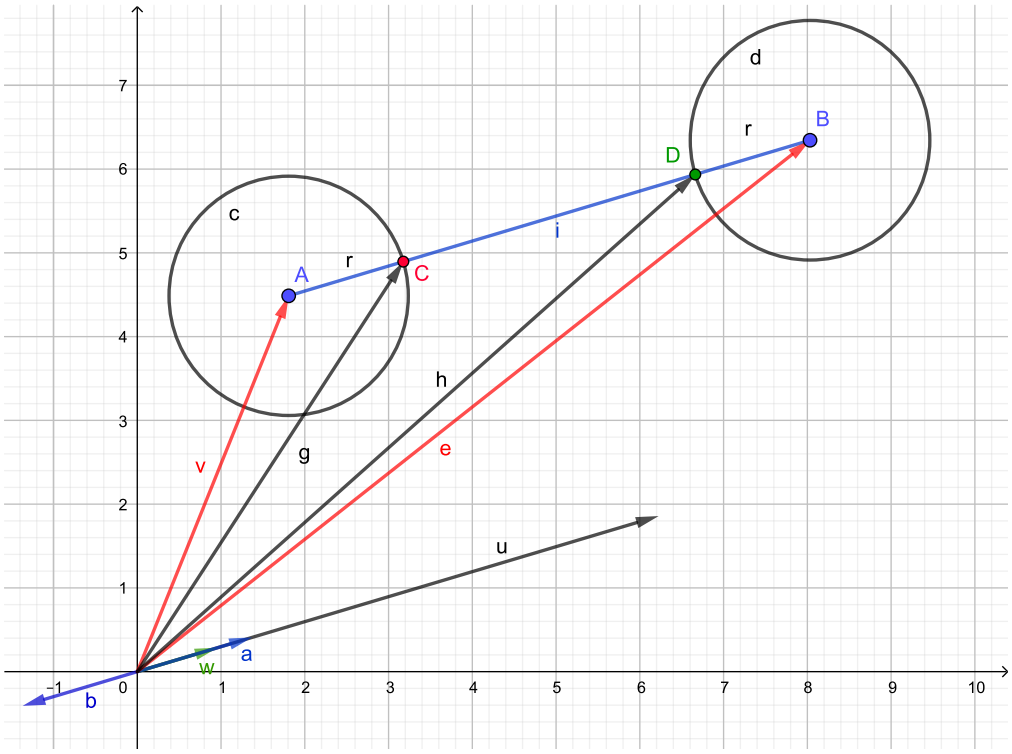
12.1.8 Rutenettet

Når en endring har skjedd på lerretet, må det tegnes på nytt. Alle noder som beveges rundt i lerretet, vil tilpasse sine koordinater etter rutenettet. Nodekoordinatene rundes av til nærmeste faktor av rutenettvariabelen som er en tallverdi som bestemmer bredden og høyden på rutene. Det er også denne variabelen som rutenettet tegnes med, og på den måten vil alle noder falle akkurat der hvor rutenettet krysser(Se figur 11).

12.2 Beregning

Siden JavaScript ikke kommer med noen vektormetoder i utgangspunktet, er metodene man behøver implementert i en egen prosjektfil kalt «vektormetoder.js».

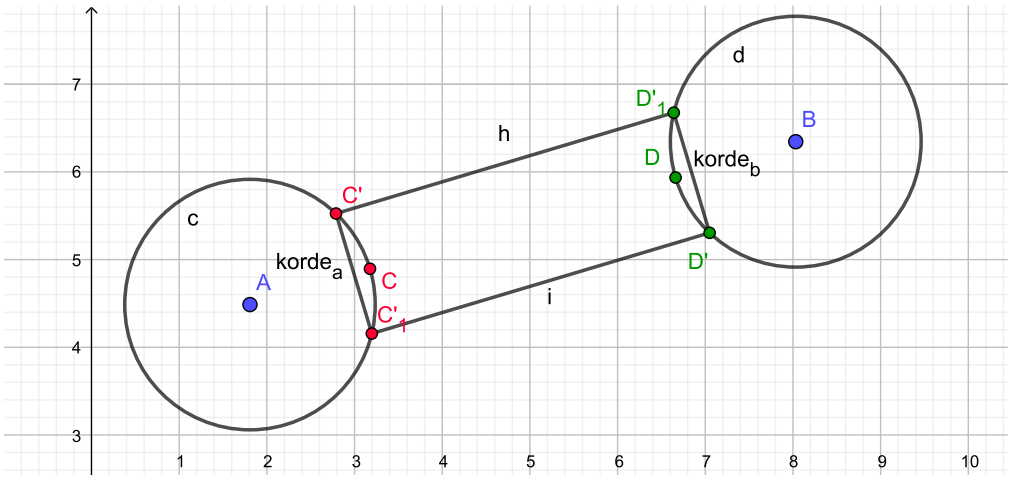
12.2.1 Noder og kanter



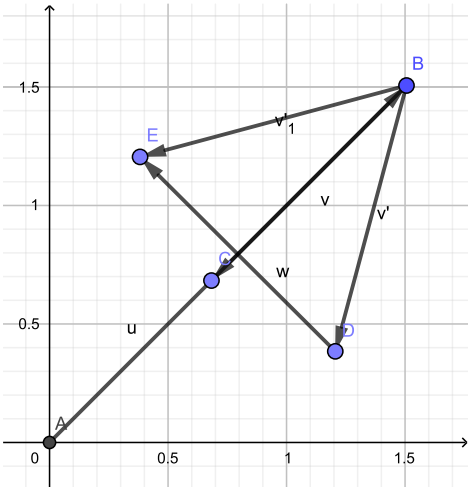
Figur 12: Beregner punkt C og D for å tegne gren i mellom nodene.

For å beregne punktene som vi skal tegne kanter mellom, kan vi ta i bruk vektormatematikk. I figur 12 ser vi hvordan dette blir gjort. Man starter med å lage en vektor mellom punktene A og B kalt u, deretter normaliser vi denne, som gir w, og skalerer w til radius r og får dermed a. Legger vi til posisjonvektoren v, får vi en vektor som går direkte til punkt C, for å finne D inverserer vi a, som gir b, og legger til posisjonvektoren til B. Ved tegning av en dobbelkant gjøres mye av de samme beregningene, bare at de to punktene på hver av nodene blir gitt ved en korde (se figur 13).

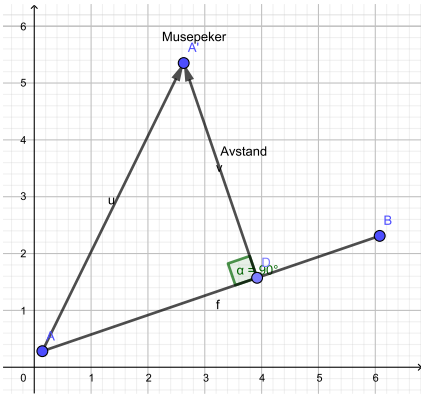
Tegning av pil er enkelt om vi bruker retningsvektoren vi fant mellom nodene, inverserer denne og skalerer med ønskelig hypotenuslengde. Deretter kan vi rotere med ønskelig vinkel f.eks. $\frac{\pi}{6}$ for å finne D, og rotere motsatt vei for å finne vektoren til E. (Se figur 14). For å registrere treff på en kant, kan man regne ut avstanden mellom musepeker og linjestykket, ved å projisere langs grenen, og så ta lengden av den ortogonale vektoren. Deretter sjekker man hvorvidt denne avstanden faller innenfor et gitt tallområde(se figur 15).



Figur 13: Bruker korder for å tegne dobbelkant.



Figur 14: Punkter for å tegne pil.



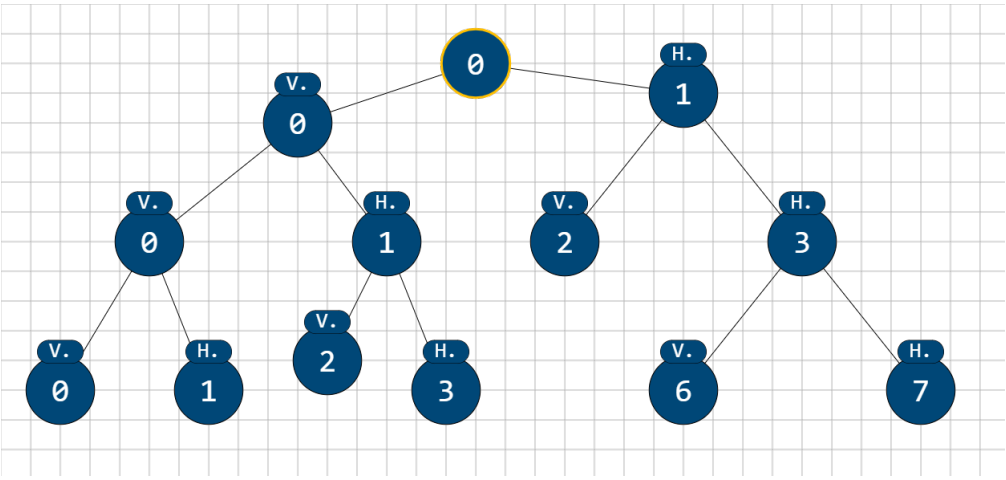
Figur 15: Avstand mellom linjestykke og musepeker.

12.3 Automatisk formatering

12.3.1 Generell formatering

Ettersom applikasjonen skal kunne brukes på store og små skjermer, trenger man å skalere lerretet og dets innhold. Når strukturtegneren åpnes, leses skjermstørrelsen av og lerretet blir tilpasset skjermen. Siden posisjonen til alle objektene er basert på den faktiske posisjonen på lerretet i piksler, ganges bare disse koordinatene med forholdet mellom den gamle og nye lerretstørrelsen.

12.3.2 Formatering av binære søketrær



Figur 16: Uformatert tre. Nøkkelen representerer nodens kolonne.

Ved formatering av et tre ønsker vi at treet skal være symmetrisk samtidig som det skal være plass til alle nodene. Deler vi lerretet inn i like mange kolonner som vi har bladnoder, kan vi få plass til alle nodene. Siden rutene i rutenettet er kvadratiske, blir det like mange ruter i bredden som i høyden.

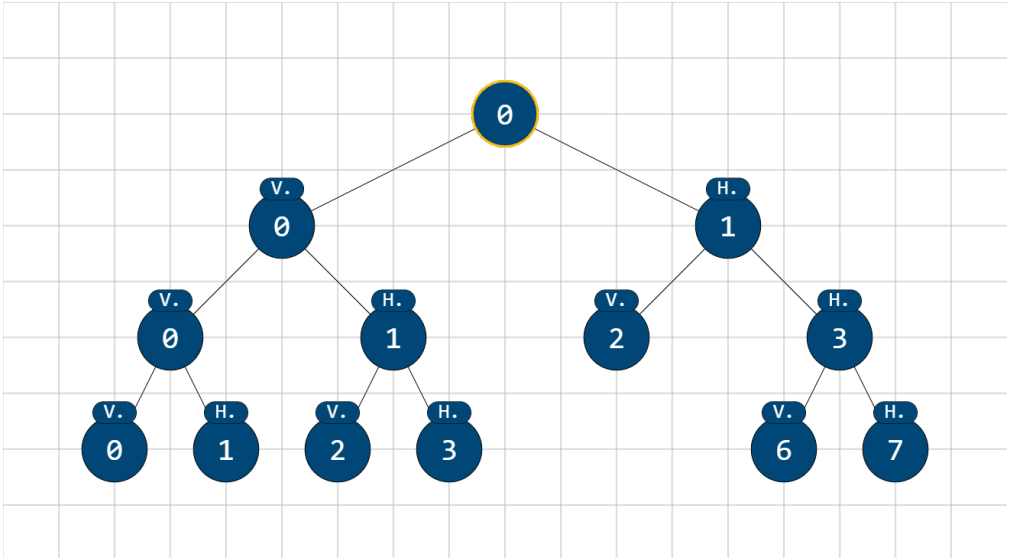
Hvis vi ser på bladnodene først kan vi tenke oss at hver enkelt bladnode skal plasseres i hver sin kolonne bestående av to ruter i bredden. Plasserer vi bladnoder midt i mellom to ruter, vet vi at om vi plasserer deres foreldre midt i mellom bladnodenes kolonner vil også forelderen plasseres midt i mellom sine egne to ruter. Siden vi vet at at det er $2^{h\phi yde}$ bladnoder, må vi dele inn i $2^{h\phi yde}$ kolonner og vi må ha dobbelt så mange ruter, det vil si $2^{h\phi yde+1}$.

Siden strukturen vi ønsker å manipulere er et tre, er det naturlig å bruke rekursjon til å utføre en generell formatering mens man traverserer hver enkel node i treet. Alternativet ville vært å bruke flere løkker. Når vi traverserer treet, begynner vi selvfølgelig på toppen ved rotnoden, og den skal bare plasseres midt på lerretet i kolonne 0.

Hvilke kolonner barna skal plasseres i, er imidlertid mer interessant, i figur 16 kan vi se at det danner seg et mønster: Helt ytterst til venstre vil alle ligge i kolonne null, men mye viktigere er det at alle venstrebarn vil ligge i forelderens kolonne ganget med to, mens alle høyrebarn vil ligge i venstrebarnets kolonne pluss en (se figur 17). Det er utgangspunktet for den rekursive funksjonen som gir oss et formatert tre i figur 18.

$$Første forelder = 0$$
$$Venstrebarn = 2 * forelder$$
$$Høyrebarn = 2 * forelder + 1$$

Figur 17:



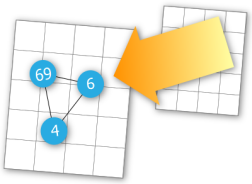
Figur 18: Formatert tre. Nøkkelen representerer nodens kolonne.

12.4 Menylinjen

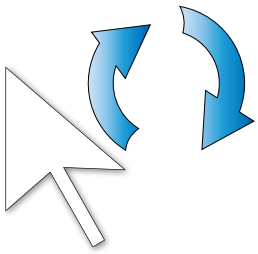
Menyen tar inspirasjon fra «ribbon menu»-grensesnittet som ble popularisert av Microsoft med Office-pakken i 2007. Menylinjen er kategorisert i tre deler: «Fil», «Verktøy» og «Visning», for å gjøre det lettere å finne fram (Se figur 22). Innenfor de ulike fanene er det ulike grupperinger av knapper som hører sammen, sortert etter den viktigste funksjonaliteten på venstre hånd. Dermed blir Strukturbyggeren mer brukervennlig samtidig som menylinjen lett kan utvides med nye kategorier og funksjonalitet. En bruker skal alltid kun ha maks ett verktøy i bruk, derfor gråes verktøyet i bruk ut (se figur 23). Når et nytt verktøy blir valgt, kalles funksjonen «velgMusepeker(musepeker)», musepekervariablen brukes deretter av koden for håndtering av klikk og musebevegelse. Fanen kan også brettes inn for å ta opp mindre plass når man ikke trenger den. For ulike oppgaver er det predefinert et sett med verktøy som er tilgjengelige for bruk, samtidig som det er noen verktøy som bare finnes for administratorbrukere.

Ikonene er laget i Adobe illustrator og er designet for å gjøre det enklere skille mellom knapper, samt gi signal om hva som vil skje om man trykker inn knappen. For å kunne formidle hva som skjer, er det opptil tre designelementer som sier noe om objekter som blir berørt og handlingen som skal utføres. Et eksempel er ikonet for å laste inn startstruktur: grønn pil = last inn, tre blå noder = struktur og hvitt rektangel = lerretet (Se figur 19).

Figur 19: Last inn struktur

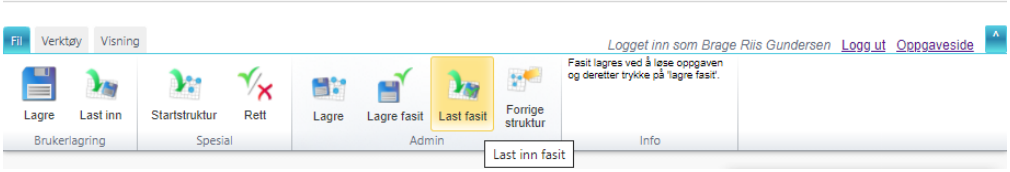


Figur 20: Last inn forrige struktur



Figur 21: Roter

12.4.1 Fil



Figur 22: Menylinjen der fil-fanen er valgt. Administrator er logget inn derfor vises «Admin» og «Info»-panelene.

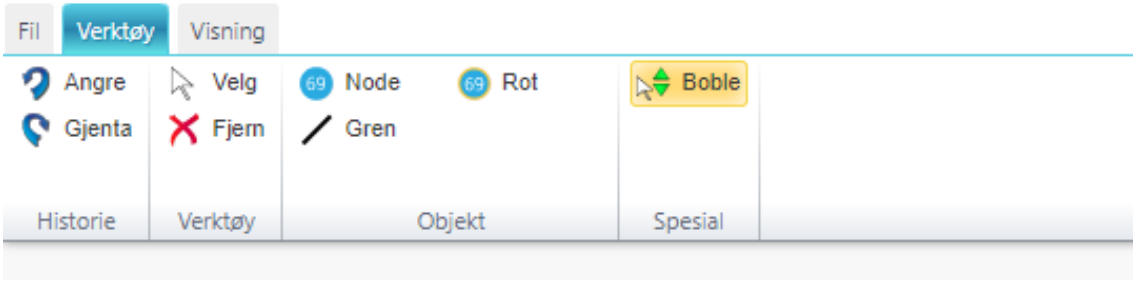
I filfanen finnes funksjonalitet som har med hvilken data som er lastet inn i Strukturbyggeren å gjøre (Se figur 22). «Lagre» lagrer det som ligger på lerretet i en mappe for brukeren på tjeneren. Brukeren kan deretter trykke «last inn» for å laste inn den lagrede strukturen.

«Startstruktur»-knappen fjerner det som ligger på lerretet og laster inn strukturen som fulgte med deloppgaven.

«Rett»-knappen retter oppgaven og gir deg med engang resultatet på skjermen.

For administratorer er det noen ekstra knapper. «Lagre» lagrer startstruktur og deloppgaven, mens «Lagre fasit» lagrer lerretet som fasit til deloppgaven. For å hente inn en struktur man trenger fra forrige deloppgave kan man trykke på «Forrige struktur».

12.4.2 Verktøy



Figur 23: Menylinjen der verktøyfanen og verktøy for å boble er valgt.

Verktøyene «angre» og «gjenta» kan brukes til å angre eller hente tilbake enkelte endringer. Plassering og fjerning av elementer som noder og kanter, kan angres og hentes tilbake.

Verktøyet «Velg» brukes til å flytte og velge en node for redigering. Når en node trykkes på sjekker programmet listene over noder og grener, og tester om musens (x, y) er innenfor radiusen til en node, eller om vektoravstanden mellom en gren og musen er mindre enn en satt lengde. Hvis brukeren trykker på et objekt, settes en global variabel til en objektreferanse som hører til objektet man trykket på. Referansen brukes i andre funksjoner for flytting og redigering av objektet.

Verktøyet «Node» brukes til å plassere noder. Når brukeren trykker på lerretet hentes musens (x, y), og en ny node legges til i «nodekartet» (dictionary struktur) med disse koordinatene (rundet av til nærmeste faktor av rutenettvariabelen). Til slutt tegnes lerretet på nytt slik at den nye noden vises for brukeren. «Rotnode»-verktøyet gjør det samme, bortsett fra at noden som lages, blir definert med et «rot»-flagg, slik at vi kan definere toppen av trær.

Verktøyene «Rettet kant», «Kant» og «Gren» brukes til å tegne kanter eller grener. Når brukeren trykker på lerretet sjekkes det om brukeren trykket på en node, slik som for «Velg»-verktøyet. I det brukeren trykker på den første noden tegner programmet en midlertidig kant til musepekerens posisjon. Dette gjøres for at brukeren skal være klar over at det tegnes en gren. Gitt at koblingen er gyldig, vil et nytt trykk på en annen node lage en ny kant som legges til i listen. Er ikke koblingen gyldig, vil brukeren få vist en melding som forklarer hvorfor grenen ikke kan kobles til. Hvis man trykker på en tom del av lerretet vil koblingen kanselleres.

≪Fjern≫ brukes til sletting av både noder og grener. Ved trykk på lerretet sjekkes det om et objekt er trykket på og objektet slettes. Ved sletting av noder må også nodens tilkoblede grener slettes.

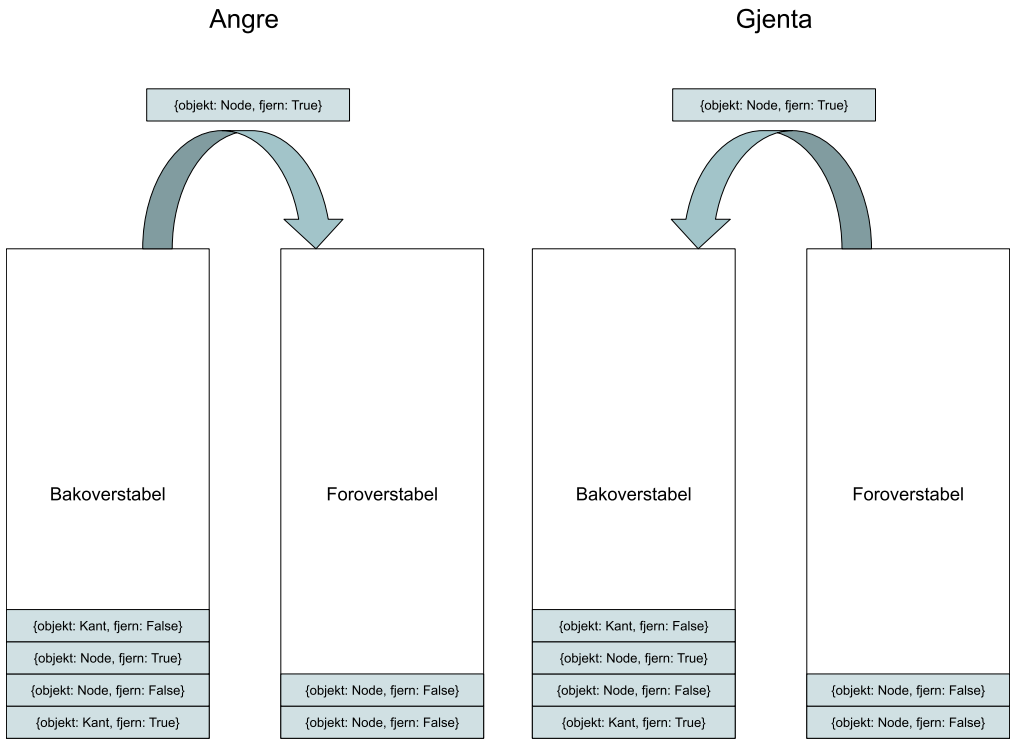
I tillegg finnes det noen spesialverktøy for bruk i spesifikke oppgavetyper:

- ≪Velg riktig≫ for oppgaver der man skal trykke på riktig nodetype.
- ≪Velg rekkefølge≫ brukes i oppgaver der man skal trykke inn rekkefølgen til ulike algoritmer: preorder, inorder, postorder og Dijkstra.
- ≪Resett rekkefølge≫ tilbakestiller rekkefølgen.
- ≪Velg rotasjon≫ brukes til å balansere AVL-trær ved å gjøre høyre- og venstrerotasjoner, der hvor rotasjonene er animert slik at man får et visuelt inntrykk av hvordan strukturen endrer seg.
- ≪Boble≫ brukes til å boble hauger og er animert.
- ≪Tegn Nodenett≫ brukes av administratorer til å tegne et nett av noder. Dette kan være nyttig i situasjoner hvor man skal lage ≪korteste vei≫-oppgaver oppå et kart.
- ≪grafmeny≫-verktøyet brukes til å visualisere grafalgoritmer.

12.4.3 Angre- og gjentafunksjonalitet

Angre- og gjentafunksjonaliteten i denne applikasjonen er implementert ved å bruke to stabler, én for nylige endringer, og én for angrede endringer. Dersom brukeren legger til et nytt element (node eller kant) eller fjerner et element, vil endringen havne i bakoverstabelen som et objekt. Dette objektet inneholder elementet og en boolsk verdi som forteller om endringen var en plassering eller en fjerning. Denne stabelen inneholder alle endringer gjort av brukeren. Dersom brukeren vil gå tilbake på endringene, kan angreknappen benyttes. Angrefunksjonen henter det øverste objektet på stabelen og legger det til på en annen stabel kalt foroverstabel. Objektet inspiseres etterpå, og dersom endringen er en plassering av et element, fjernes elementet fra lerretet. Dersom endringen er fjerning av et element, legges elementet tilbake på lerretet.

Brukeren kan også velge å gjenta en endring dersom brukeren vil ha endringen tilbake. Dette gjøres ved å hente det øverste elementet fra foroverstabelen og legge det tilbake på foroverstabelen.



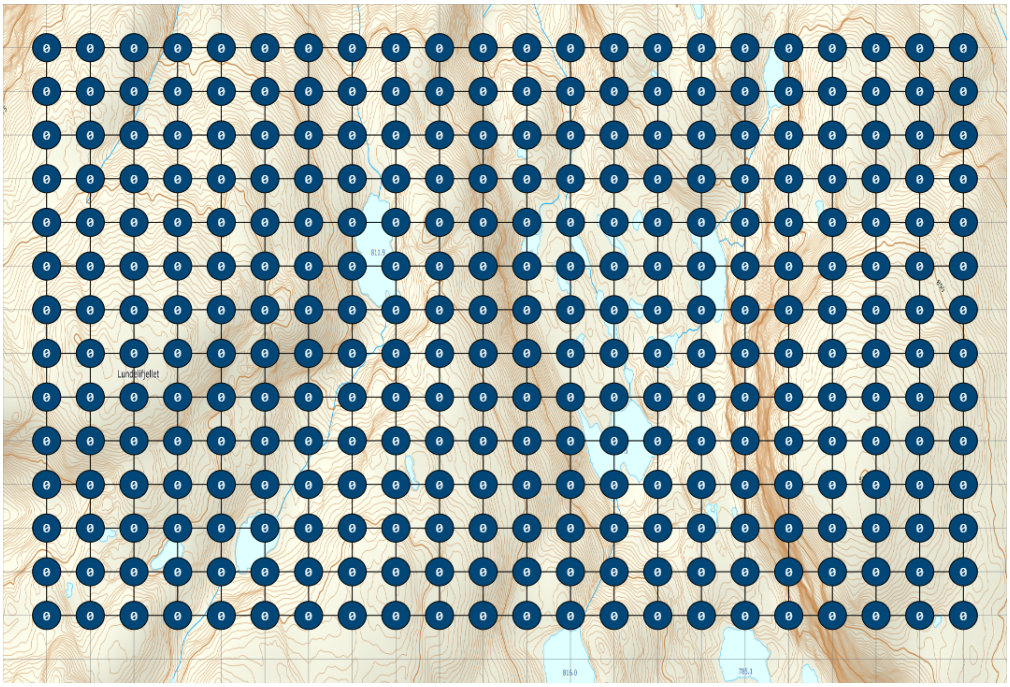
Figur 24: Bakover- og foroverstabel.

Dersom flere enn ti elementer legges til i en av stablene, vil elementet nederst i stabelen bli fjernet. Dette gjør at man maksimalt kan bevege seg ti endringer i begge retninger.

For å sikre at angre- og gjentafunksjonaliteten fungerer som forventet, tømmes foroverstabelen dersom brukeren gjør en ny endring på lerretet. Dermed vil man ikke kunne gjenta endringer som blir ugyldige etter at man gjør en ny endring.

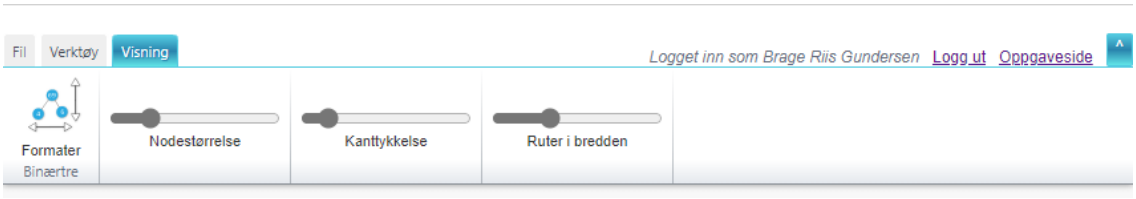
12.4.4 Nodenett

For å tegne et rutenett av noder, brukes applikasjonens rutenett. Ved å beregne antall ruter i høyde og bredde, kan noder plasseres jevnt rundt på lerretet.



Figur 25: Eksempel på et automatisk generert nodenett.

12.4.5 Visning



Figur 26: Menylinjen der visningfanen er valgt.

Visning-menyen inneholder all funksjonalitet som endrer hvordan strukturer blir vist. Her er det mulig å endre hvor store noder er, eller hvor tykke kantene er. Man kan også endre antall ruter i bredden rutenettet har. I tillegg er det implementert et verktøy for å automatisk formatere trær.

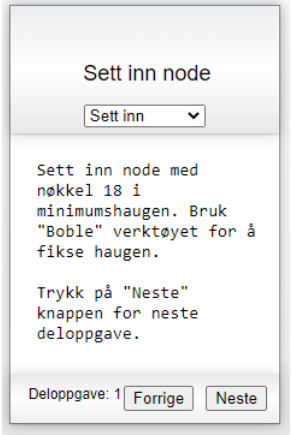
13 Animasjon

For å animere en node som skal flyttes, implementerte vi en funksjon som tar inn en node, og den nye (x, y) posisjonen til noden. For å finne ut hvor mye hvert flytt skal bevege seg i x og y, trekkes den nye x-posisjonen fra nodens nåværende-x posisjon. Dette gjøres også for y. Disse differansene deles på antall steg animasjonen skal ha. Ettersom mange noder ofte skal flyttes samtidig, er det nødvendig å redusere antall ganger lerretet tegnes. Derfor vil ikke rutenettet bli tegnet på nytt for hver node som skal flyttes, men heller én gang for alle nodene som skal flyttes.

14 Oppgaver

14.1 Lage deloppgaver

Når en administrator skal lage en ny deloppgave, kan deloppgavetypen velges fra menyen til høyre på skjermen. Ved å trykke «Lagre» i «Fil»-menyen, lagres den valgte deloppgavetypen, tittel, beskrivelse og strukturen som er tegnet på lerretet.

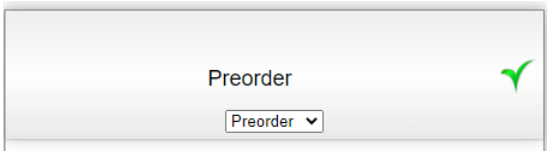


Figur 27: Meny for laging og velging av deloppgave.

«Neste» og «Forrige»-knappene kan brukes av både administratorbrukere og brukere til å forflytte seg mellom deloppgaver, men bare administratorer vil kunne lage en ny deloppgave.

Figur 28: En administrator som har valgt en ny deloppgave. «(ny)» betyr at deloppgaven er ny og ikke er lagret.

Når en deloppgave har blitt løst av en bruker, vil en grønn avhakning vises.



Figur 29: En bruker har svart riktig på en deloppgave. Visualiseres med en grønn sjekk.

14.2 Binært søketre

≪Tegn binærtre≫-oppgavetypen er laget med tanke på at man som bruker skal bygge et binært søketre ut ifra gitte kriterier. I oppgaven er det bare tillatt å plassere noder og grener. Det er viktig at oppgavebeskrivelsen definerer en liste med noder i den bestemte rekkefølgen de skal plassers i, slik at det kun finnes en løsning. Deretter lager administratoren en fasit og lagrer den. Treet kan formateres med ≪formater≫-verktøyet.

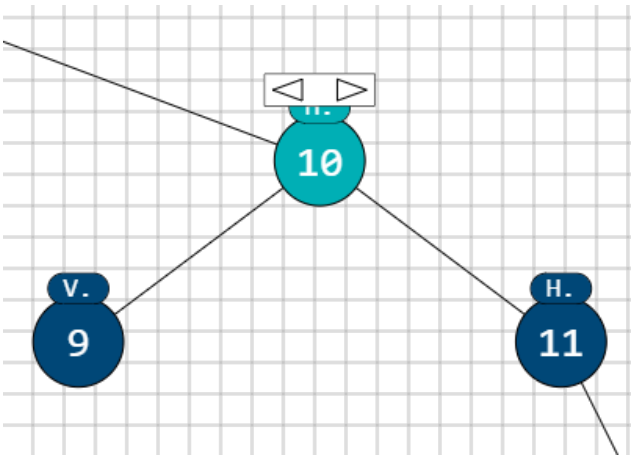
14.3 Velg riktig

≪Velg riktig≫-oppgaven går ut på at man enkelt og greit skal trykke på de ulike typene noder i treet for å teste om brukeren vet hva de ulike nodene i treet kalles. Man velger enten rotnoden, bladnodene eller de vanlige nodene.

14.4 Velg rekkefølge

I ≪Velg rekkefølge≫-oppgaven er meningen at man skal teste sine egne ferdigheter når det gjelder tretraversering. Det er lagt inn tre forskjellige algoritmer man skal prøve å etterligne: inorder, preorder og postorder.

14.5 AVL-trær



Figur 30: Grensesnitt for å rotere tre

14.5.1 Roter AVL-tre

Det kan være vanskelig å se for seg hvordan man kan utføre rotasjoner i AVL-trær, og oppgaven prøver å hjelpe til med dette. I denne deloppgaven er det definert et tre som skal balanseres slik at treet oppfyller kravene for å være et AVL-tre. Definisjonen på et AVL-tre er at alle venstrebarn har nøkkel mindre en forelderen og alle høyrebarn har større nøkkel enn forelderen, i tillegg til at det maksimalt kan være én i høydeforskjell mellom bladnoder. For å balansere treet er det implementert et grensesnitt inne i Strukturbyggeren som viser seg når man trykker på en av nodene. Brukeren trykker så på enten høyre eller venstrepil for å rotere. Kalkulasjonen som gjøres for å bestemme hvordan treet skal endre seg etter hver rotasjon, er relativt innviklet, og det er mange kontroller som må gjøres før rotasjonen kan bli animert på lerretet.

14.5.2 Tegn AVL-tre

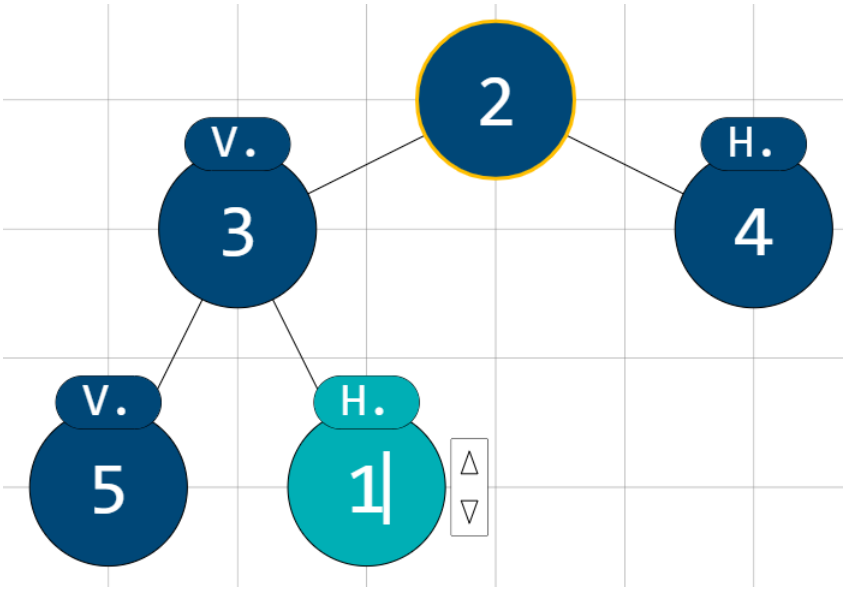
Denne deloppgaven går ut på at man skal bygge et AVL-tre ut i fra et sett med noder som er spesifisert i oppgavebeskrivelsen. Underveis i byggeprosessen skal man rotere treet for at det skal bli et gyldig AVL-tre.

14.6 Haugoppgaver

I denne applikasjonen skal brukeren kunne løse enkelte oppgaver rundt maksimums- og minimumshauger. En minimumshaug lagrer de minste nøklene i toppen av treet, mens en maksimumshaug lagrer de største verdiene i toppen av treet. Dette gjør at oppgavene for de to haugene blir relativt like.

14.6.1 Sett inn

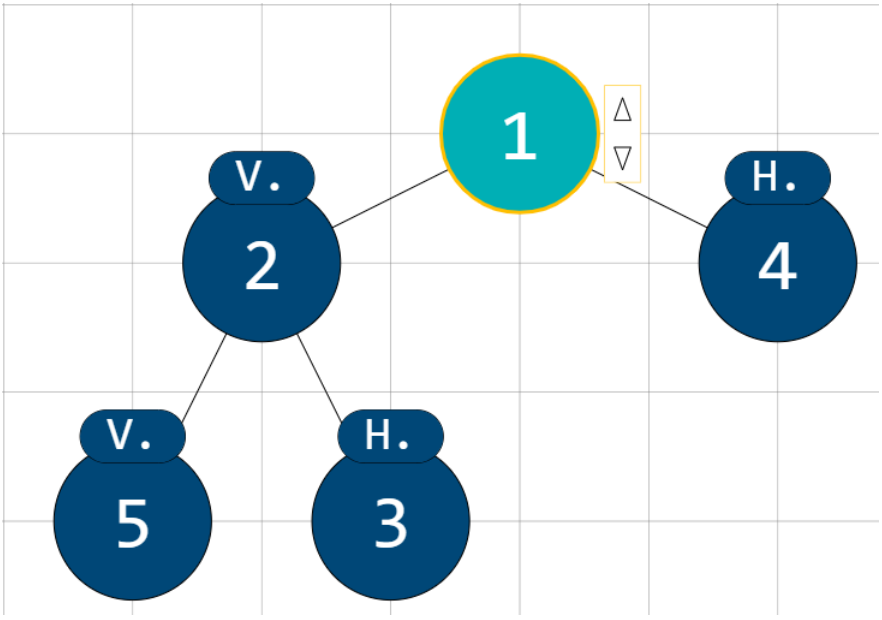
En av deloppgavetypene vi implementerte for hauger var «sett inn»-funksjonen. Når man setter en ny node inn i en haug, setter man noden inn i bunnen av haugen og så langt til venstre som mulig. Noden må deretter flyttes opp til den ligger på riktig sted. Dette kalles bobling. I en minimumshaug må man boble opp dersom nodens nøkkel er mindre enn nøkkelen til noden ovenfor, for en maksimumshaug er det motsatt. For å gjøre denne prosessen lettere og mer intuitiv for brukeren, implementerte vi bobleverktøyet.



Figur 31: Ny node med nøkkel 1 valgt med bobleverktøyet.

Bobleverktøyet er et verktøy som automatisk bobler en gitt node. Ved å bruke verktøyet trenger ikke brukeren å fjerne eller koble grener eller manuelt flytte noder. Verktøyet sørger for at nodene som flyttes blir gitt riktig foreldernode-ID, og riktige barnnode-ID. Nodene flyttes også med bruk av animasjon slik at brukeren ser hvordan nodene beveger seg. Eventuelt kunne bobleverktøyet blitt implementert ved å bare endre nodevekten mellom de to nodene som skal byttes, men dette ville ikke animert nodene, og nodenes nøkler ville bli koblet til en annen node-ID.

En typisk «sett inn»-deloppgave kan være at brukeren skal utvide en eksisterende haug ved å sette inn noder. Gitt at brukeren kan prosessen om hvordan noder plasseres inn i hauger, vil han kunne plassere noden på riktig sted og boble den opp til riktig plass.



Figur 32: Node med nøkkel 1 boblet to ganger.

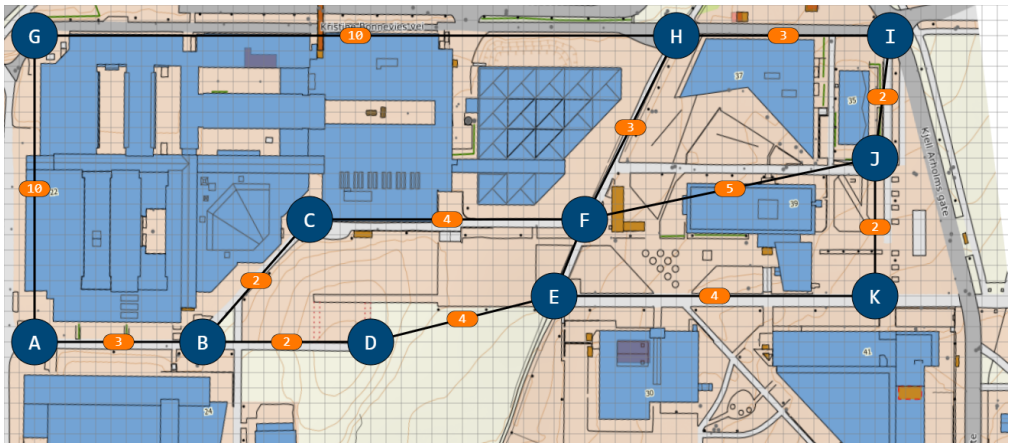
14.6.2 Pop fra haug

En annen haugoperasjon vi implementerte som en deloppgave, var fjerning av øverste element i en haug. En typisk deloppgave vil være at brukeren skal fjerne den øverste noden. I denne deloppgaven kan også bobleverktøyet brukes.

14.6.3 Bygg haug

I denne deloppgavetypen er tanken at brukeren skal bli gitt et sett med nøkler som skal plasseres inn i strukturbyggeren for å bygge en haug på riktig måte.

14.7 Graf



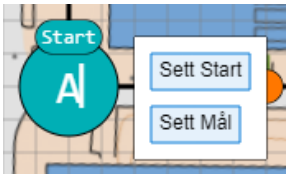
Figur 33: Graf

Grafoppgavene er tenkt mest som visualisering og lek med algoritmene, men i tillegg er det implementert deloppgaver hvor man velger veien disse algoritmene tar.

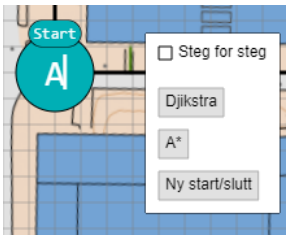
14.7.1 Grensesnitt

For å visualisere algoritmene er det implementert ett eget grensesnitt for å gå gjennom algoritmene i lerretet. Grensesnittet tegnes inne i lerretet, og først av alt får man mulighet til å sette start og mål for algoritmen når man trykker på en av nodene(figur 34). Etter at starten og målet er definert, kan man bruke menyen i figur 17 for å starte en av algoritmene, og eventuelt velge å gå gjennom algoritmen steg for steg med menyen i figur 36. Etter at algoritmen er kjørt vises den beste stien som ble funnet i grønn slik som i figur 37. Hver av menyene er bygget opp ved hjelp av klasser med sine egne strukturer og måter å tegne på. Menyobjektet inneholder knappeobjektet og annet som skal tegnes på menyen, og som skal kunne trykkes på. Menyen i seg selv regner også ut sin egen størrelse og passer på at den ikke havner over den valgte noden eller utenfor lerretet.

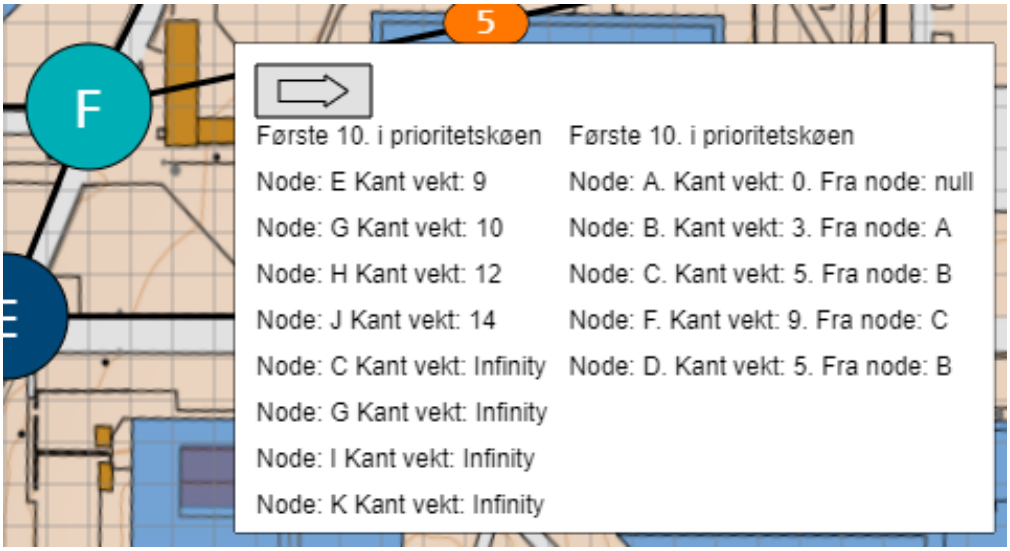
Et alternativ til den integrerte menyen kunne være å lage et eget vindu eller integrert menyen utenfor lerretet, men grafmenyen viser at det også er mulig å lage et grensesnitt som fungerer uten at man må bevege seg ut av lerretet.



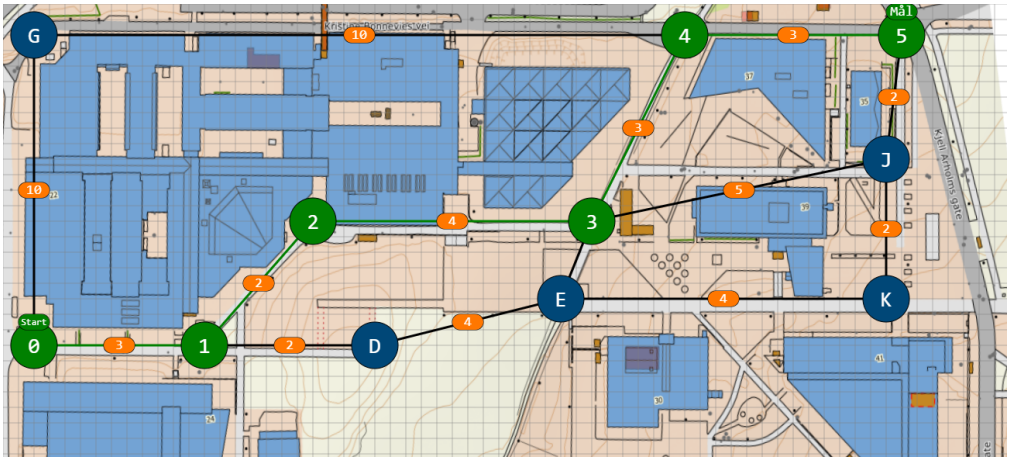
Figur 34: Punkter for å tegne pil.



Figur 35: Avstand mellom linjestykke og musepeker.



Figur 36: Graf



Figur 37: Sti for Dijkstra-algoritmen

14.7.2 Velg Dijkstras-algoritme-rekkefølge

Denne deloppgavetypen lagde vi for å teste brukerens forståelse av Dijkstras «korteste vei»-algoritme. Dijkstras algoritme finner den korteste veien mellom en start- og en sluttnode, ved hjelp av en prioritetskø. I denne prioritetskøen legger algoritmen grafnodene og sorterer dem etter lavest kumulativ kantvekt fra startnoden, men lagrer også «fra-noden» som ble brukt til å sette vekten. Algoritmen går stegvis gjennom nodene i prioritetskøen som har lavest vekt, og oppdaterer kumulative kantvekter og «fra-noder» til nabonoder underveis. Ferdige noder legges i en egen liste, og noden fjernes fra prioritetskøen. Algoritmen fortsetter til prioritetskøen er tom. For å finne den korteste veien kan algoritmen gå til sluttnoden, og traversere gjennom nodene («fra-nodene») som ble bruk til å sette kantvekten (Computerphile, 2017, 2:46).

Ettersom algoritmen har et kart som inneholder alle besøkte noder, kan man lage en oppgavetype som går ut på å velge den rekkefølgen algoritmen skal besøke nodene i. En typisk deloppgave vil være at brukeren får en ferdiglaget graf som inneholder flere kanter med ulike vekter. Brukeren får også oppgitt start- og sluttnoden. Brukeren skal da se på grafen, og bruke «Velg rekkefølge»-verktøyet til å velge rekkefølgen algoritmen vil besøke de ulike nodene i grafen i.

Dersom det finnes flere veier mellom start- og sluttnoden med samme vekt, vil fortsatt bare en av veiene gi riktig svar. Dette er fordi oppgaven går ut på å velge rekkefølgen algoritmen vil besøke nodene. I denne applikasjonen er nodene lagret i en liste. Når algoritmen henter nabonoder vil dermed nodene som ligger foran i listen bli hentet ut først og lagt i prioritetskøen. Skulle to eller flere naboer ha samme vekt, vil da noden som ligger først i listen også havne først i prioritetskøen. Dersom de andre nodene ikke får en lavere vekt vil da denne noden besøkes først av algoritmen. Det er derfor lurt å informere om dette i oppgavebeskrivelsen.

15 Klientsidens kodestruktur

For å holde koden strukturert delte vi klientsidekoden i applikasjonen i flere filer:

- «angre.js» inneholder kode som gjør det mulig å angre og gjenta endringer gjort i lerretet.
- «brukerdataLagring.js» inneholder kode som gjør det mulig for brukere å lagre sin egen data.
- «grafAlgoritmer.js» inneholder kode for henting av grafalgoritmedata fra tjener og visualisering av dette på lerretet.
- «kant.js» inneholder Kantklassen og kode som omhandler kanter.
- «konfigurer.js» inneholder kode som håndterer endringer i strukturbyggeren. Dette gjelder blant annet endring av nodenøkkel, endring av deloppgave og diverse menyfunksjonalitet.
- «lagring.js» inneholder koden for å lagre og hente data fra tjeneren.
- «mus.js» inneholder kode for håndtering av valgt verktøy og muse-hendelser.
- «node.js» inneholder Nodeklassen, og håndterer viktige nodefunksjoner.
- «retting.js» inneholder kode for henting av riktig svar fra tjeneren, og visualisering av eventuelle feil.
- «tegn.js» inneholder koden som tegner elementer på lerretet.
- «script.js» inneholder kode som ikke passer inn i de andre filene.
- «vektorMetoder.js» inneholder implementasjonen for diverse vektormetoder som er viktig for tegning av noder og kanter.

16 Tjenerside

16.1 Oppstart

For å starte serveren må diverse biblioteker installeres. Bibliotekene kan bli installert ved hjelp av Pythons sitt «pip»-verktøy i kommandolinjen. Disse kommandoene befinner seg i «requirements.txt»-filen.

16.2 Tjenersidens kodestruktur

Som for klientsidekoden delte vi tjenersidekoden i flere filer:

- «__init__.py» kode som initialiserer tjeneren.
- «brukerdataLagring.py» inneholder rutene for å hente og lagre brukerdata.
- «commands.py» inneholder kommandoene som kan skrives i tjenerens terminal for å hente ut nyttig databasedata.
- «db.py» inneholder kode for initialisering og henting av databasen.
- «grafFunksjoner.py» inneholder funksjoner og ruter relatert til grafalgoritmer.
- «lagring.py» inneholder ruter og funksjoner for lagring av oppgavesett, oppgaver, deloppgaver og fasit.
- «login.py» inneholder kode relatert til innlogging ved hjelp av Feide og Flask-login.
- «main.py» inneholder ruter til applikasjonens hovedsider.
- «retting.py» inneholder funksjoner og ruter for retting av deloppgaver.
- «sqloperasjoner.py» inneholder alle SQLite-spørringer for henting og lagring av brukerdata i databasen.

16.3 Ruter og tilgang

Applikasjonen har to hovedruter. ≪/≫ er ruten til innloggingssiden og oppgavesiden. Hvilken side avhenger om brukeren er logget inn eller ikke. ≪/strukturbygger≫ er den ruten til strukturbyggeren. ≪/≫ er den eneste ruten som kan besøkes uten innlogging.

Enkelte ruter er bare tilgjengelige dersom brukeren er en administrator. Dette gjelder følgende ruter:

- ≪/visOversikt≫: Henter administratoroversikten.
- ≪/lastInn≫: Tilgjengelig for alle brukere dersom en oppgave skal lastes inn. Dersom en fasit skal lastes inn må brukeren være en administrator.
- ≪/lagre≫: Lagrer en ny deloppgave.
- ≪/lagreFasit≫.
- ≪/lagreOppgavesett≫.
- ≪/nyOppgave≫.
- ≪/fjernOppgave≫.
- ≪/fjernOppgavesett≫.

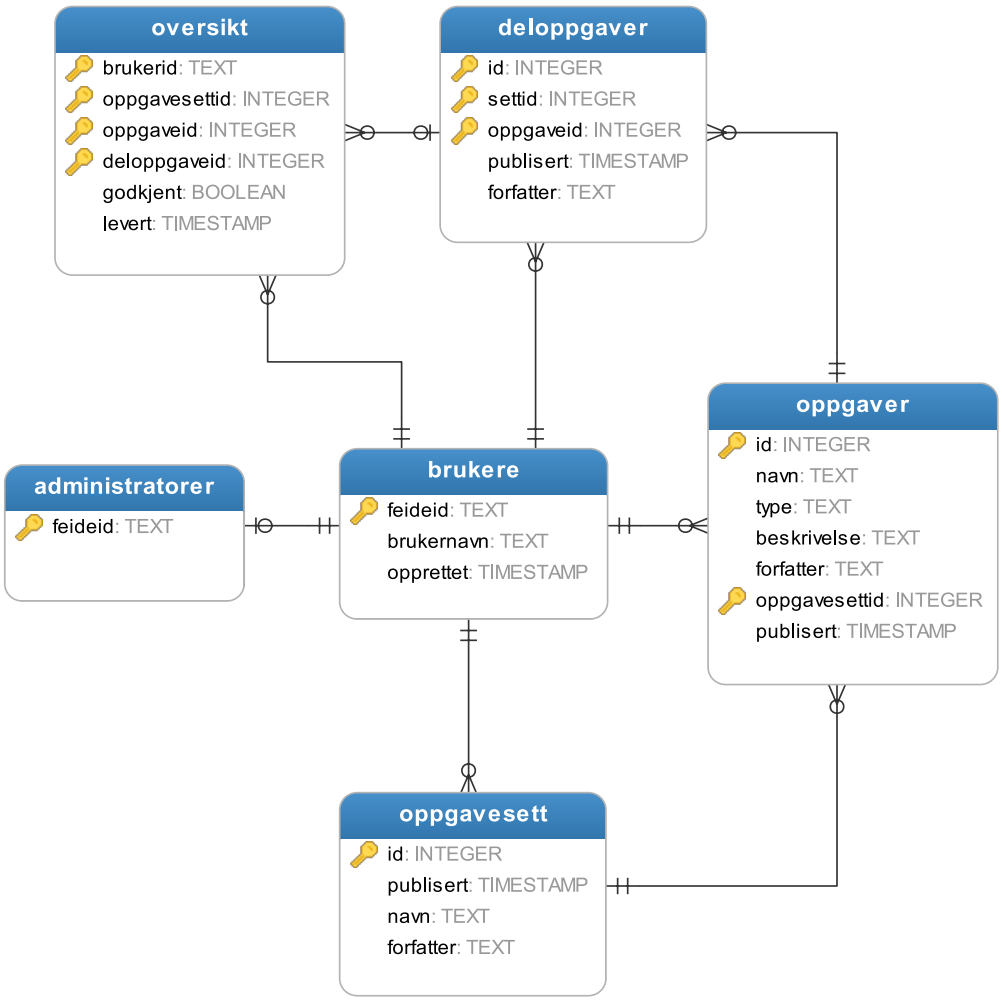
16.4 Database

Databasen er implementert i SQLite3 som er integrert i Flask, og isteden for at det er en databaseprosess som kjører på tjeneren til enhver tid, er det applikasjonen som manipulerer databasen direkte. Dette er enklere, raskere (i mindre applikasjoner) og krever mindre konfigurering. Databasefilen blir låst hver gang det skjer en skriveoperasjon, mens leseoperasjoner kan skje samtidig. Dermed blir det ikke noe problem med redusert ytelse ved stor pågang og mange samtidige skriveoperasjoner, slik det blir med andre løsninger, der operasjonene må skje sekvensielt.

Databasen består av seks tabeller (Se figur 38). Om brukere finnes i administratortabellen, anses de som administratorer. Oppgavesett, oppgaver og deloppgaver følger den naturlige formen der oppgavesettet har en eller flere oppgaver osv. Nye rader blir lagt til i oversikttabellen hver gang noen får godkjent en deloppgave og det er denne tabellen vi kan se på oversiktsiden.

For å kunne administrere tjeneren og databasen er det implementert noen kommandoer for kommandolinjen, disse ligger i «commands.py». Databaseoperasjonene som brukes i kommandoene og i resten av applikasjonen finnes i «sqloperasjoner.py».

- «initdb» - Sletter hele databasen og bygger den på nytt. Sletter alle brukerdata, oppgavesett, oppgaver og deloppgaver. (Vær varsom ved bruk av denne)
- «listbrukere» - Lister opp alle brukere i databasen.
- «listadministratorer» - Lister opp alle administratorer i databasen.
- «settadmin» - Registrerer en bruker-ID som administrator - kan være lurt å bruke «listbrukere» for å finne en bruker-ID først.



Figur 38: Databasediagram

16.5 Innlogging og brukerrettigheter

Som beskrevet tidligere var et innloggingssystem nødvendig for identifisering av brukere. Feide-tjenesten og Python-biblioteket Flask-login brukes for dette.

Når vi registrerte applikasjonen for bruk av Feide-tjenesten, kunne vi velge mellom forskjellige «scopes», som er hvilke brukerdata applikasjonen skal få tilgang til. Ettersom det er anbefalt å bare velge nødvendige «scopes» og vi bare trengte bruker-ID og navn, valgte vi bare de mest grunnleggende, nemlig «openid», «userid» og «profile».

Som en del av å implementasjonen av Flask-login i applikasjonen, måtte vi implementere en brukerklasse. Brukerklassen brukes til å holde informasjon om brukere, som brukernavn, bruker-ID og om brukeren er autentisert eller ikke. Ved å kalle *flask_login.login_user(brukeren)*, vil Flask-login gi brukeren tilgang til rutene i applikasjonen markert med *@login_required*. *flask_login.logout_user()* kalles for å logge ut brukeren. Hver gang en bruker logger inn i applikasjonen via Feide, lages et nytt brukerobjekt for den autentiserte brukeren, og brukerens ID lagres i økten. Bruker-ID lagret i økten brukes til å laste inn brukerobjektet (Countryman, 2019). Brukerobjektet inneholder brukerens ID, navn og en boolsk verdi kalt «visAdminFunksjonalitet» som gjør at diverse menyer og knapper i strukturbyggeren vises dersom verdien er *True.*

For å sjekke om brukeren er logget inn brukes *current_user.is_authenticated,* som henter ut brukerobjektet knyttet til en gitt bruker-ID og returnerer brukerobjektets «is_authenticated»-verdi. Har ikke brukeren et brukerobjekt (ikke logget inn), vil brukeren omdirigeres til innloggingssiden (Countryman, 2019).

16.5.1 Tilgang til administratorfunksjonalitet

Administratorfunksjonalitet inkluderer laging av oppgavesett, oppgaver, deloppgaver og visning av brukeroversikt. For at en bruker skal ha tilgang til disse funksjonalitetene, må brukerens ID befinne seg i administratortabellen. Hvis en vanlig bruker kaller på en administratorrute, vil brukerens ID testes mot administratortabellen, og finnes ikke ID-en i tabellen, blir brukeren nektet tilgang.

16.6 Retting av oppgaver

16.6.1 Oppbygning av rettefunksjonalitet

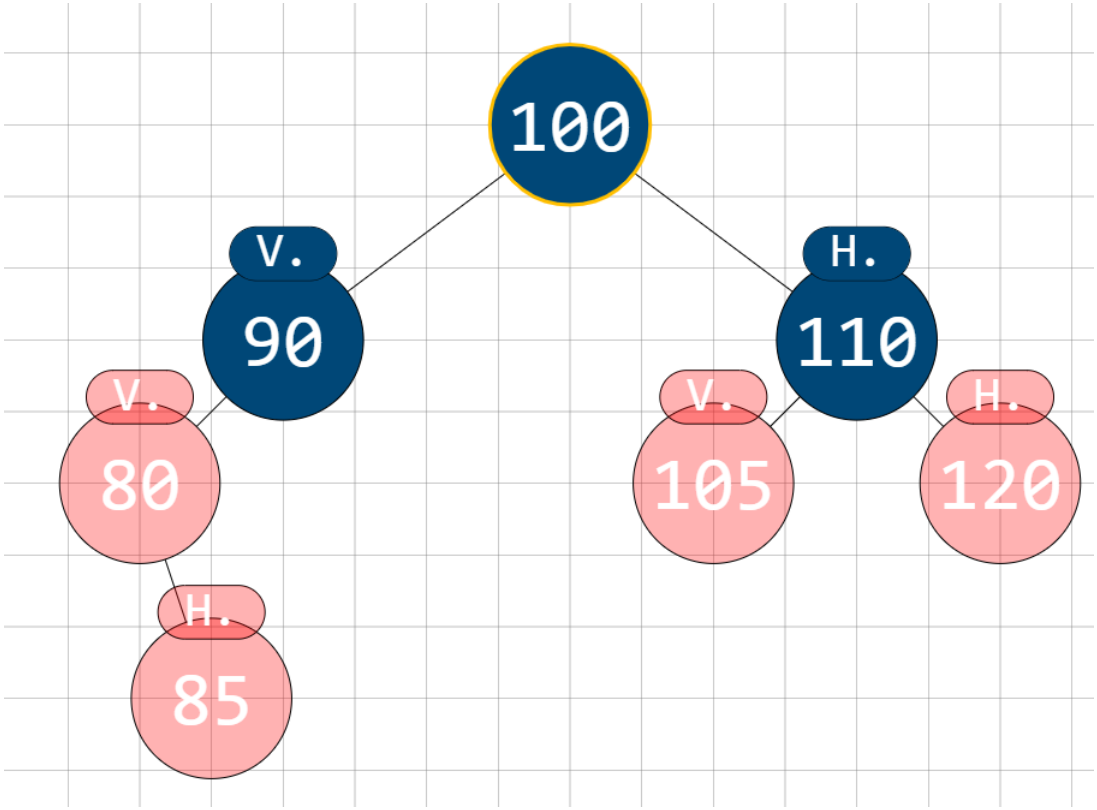
For å rette oppgaver implementerte vi funksjonen «rettOppgave», som sender brukerens strukturtegning til serveren via en «POST»-forespørsel. Hos tjeneren sjekkes oppgavetypen, og riktig rettefunksjon kjøres basert på dette.

16.6.2 Retting av trestrukturoppgave

Ettersom flere av oppgavetypene vi lagde i applikasjonen bruker binære søketrær, bestemte vi oss for å lage en generell funksjon for retting av disse oppgavetypene, som også kan visualisere feil. Rettingen av trestrukturen baserer seg på rekursiv tretraversering ut ifra rotnoden. Hvis brukerens struktur ikke inneholder en rotnode, teller det som feil. Dersom fasit ikke eksisterer eller ikke har en rotnode, vil ikke brukerens oppgave bli rettet.

For å sammenligne en innlevert struktur og fasit, implementerte vi funksjonen «finnFeil», som kalles i «rettTreOppgave»-funksjonen. Funksjonen gjør en rekursiv traversering av både fasit og brukerens struktur samtidig og sammenligner vekt og barnnoder. Dersom en barnnode mangler, lagres den manglende barnnoden i et kart med foreldernodens id som nøkkel. Man må også sjekke om barnnoden utgjør et tre, ettersom nodene i dette treet også vil mangle. Disse hentes ut fra fasit ved hjelp av en rekursiv funksjon og lagres også i kartet. Kartet returneres etter funksjonen har traversert over alle nodene i fasit og sammenlignet disse med den leverte strukturen. Hvis ingen feil ble funnet blir «Riktig» vist i grønn tekst i strukturbyggeren. Dersom brukeren har gjort én eller flere feil i oppgaven, returneres en boolsk verdi satt til «False» sammen med kartet over feil i JSON-format. «Feil» blir vist i rød tekst og feilene visualiseres.

For å visualisere eventuelle feil med brukerens struktur, implementerte vi «visualiserFeil(). Denne funksjonen bruker kartet som man har fått av tjeneren, til å visualisere hvilke noder som eventuelt har feil vekt, eller mangler høyre- eller venstrebarn. Noder som inneholder feil i forhold til fasit blir farget røde. Ved feil nodevekt settes vekten lik den riktige vekten. Mangler en node høyre- og eller venstrebarn, tegnes disse de med rød farge. Dersom en av disse barnnodene har høyre- og eller venstrebarn, vil disse også være lagret i kartet, og tegnes inn ved å rekursivt traversere ned de manglende nodene.



Figur 39: Visualisering av feil i en struktur.

16.6.3 Retting av rekkefølgeoppgaver og «velg riktig»-oppgaver

I rekkefølgeoppgavene genereres en rekkefølge på tjenersiden som brukerens rekkefølge sammenlignes med.

Retting av «velg riktig»-oppgaver fungerer på lignende måte. Brukeren sender inn de valgte nodene, og disse sjekkes mot fasit. I denne typen oppgave har ikke rekkefølgen noen betydning, så tjeneren sjekker bare om brukeren har valgt alle nodene som ligger i fasit.

16.6.4 Retting av AVL-treoppgaver

Dersom deloppgaven er å tegne et AVL-tre, sjekker applikasjonen først om høydeforskjellen i rotnodens venstre- og høyretre er større enn 1 for å se om treet er et AVL-tre. Deretter sammenlignes antall noder og kanter i fasit mot antall noder og kanter i den innleverte strukturen. Etter dette sjekkes det at den innleverte strukturen inneholder de samme nodevektene som fasit. Tilslutt sjekkes det om den innleverte strukturen oppfyller kriteriene for et binært søketre. Dersom alle sjekkene er vellykkede, blir «Riktig» sendt tilbake til brukeren.

Hvis deloppgaven går ut på å fikse et eksisterende tre med rotasjonsverktøyet, vil oppgaven bli rettet uten at en fasit er nødvendig. I stedet for å sammenligne mot en fasit, testes brukerens innsendte struktur med «sjekkOmGyldigBinærtre» og «finnHøyde» funksjonene. I «sjekkOmGyldigBinærtre» traverseres treet rekursivt, og det sjekkes at venstrebarn har lavere verdi enn foreldernodene og at høyrebarn er større. Dersom treet blir godkjent som et lovlig binært søketre, kjøres «finnHøyde» på venstre og høyrebarnet for å finne høydeforskjellen. Er forskjellen 1 eller 0, godkjennes treet som et lovlig AVL-tre. Resultatet av rettingen blir til slutt sendt tilbake til brukeren.

16.6.5 Retting av haugoppgaver

Retting av haugoppgaver sammenligner levert svar med fasit, og visualiserer eventuelle feil på samme måte som forklart i avsnittet om retting av binære søketrær.

16.7 Lagring

For at en bruker skal kunne hente oppgaver laget av en administratorbruker, trengs et lagring- og innlastingssystem. Lagringen håndteres av tjeneren som lagrer oppgavene i minnet og i JSON-format på disk. Dersom en bruker har administratorrettigheter, vil brukeren blant annet kunne lage nye oppgaver, mens en vanlig bruker bare kan laste inn oppgaver, eller lagre og laste inn sine egne brukerdata. Hver gang serveren starter på nytt, lastes oppgavene og oppgavenes fasit fra JSON-filer inn i minnet. Dette gjør at henting av lagret data ikke krever innlasting av JSON-filene flere ganger.

16.7.1 Lagringsstruktur

Dataene i denne applikasjonen er i stor grad JavaScript-objekter som noder, grener og kanter, og lagres dermed i JSON-format, som virker bra til lagring av objekter. Oppgavestrukturdata er delt inn i to filer, «oppgaver.json» og «oppgaverFasit.json», slik at oppgaver og fasiter lagres separat. Disse filene inneholder ett oppgavesett. Dette er en struktur hvor alle oppgavene handler om den samme datastrukturtypen. Et oppgavesett kan for eksempel inneholde oppgaver for minimums- og maksimumshauger, dersom oppgavesettet lagrer haugoppgaver. I tillegg til å lagre ett sett med oppgaver, lagres også en tittel som er med på å beskrive oppgavetypen settet inneholder.

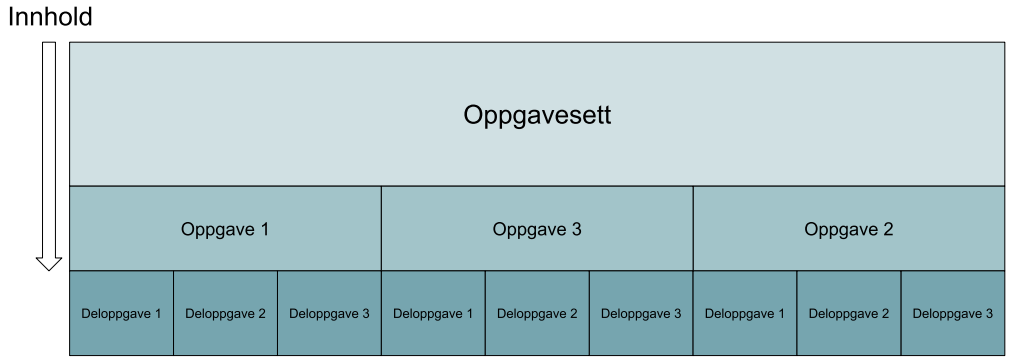
Oppgavesett inneholder som beskrevet flere oppgaver, og disse oppgavene lagrer data som tittel, beskrivelse og type. Dette skal hjelpe brukeren til å forstå hva en oppgave dreier seg om. I tillegg til dette, inneholder en oppgave også et sett med deloppgaver. Se figur 40 for en oversikt over hvordan JSON-dataene er strukturert.

Deloppgaver er oppgavene brukeren skal løse. Deloppgaven inneholder blant annet noder og kanter, som skal vises på lerretet i tillegg til oppgavebeskrivelse.

For at alle nye oppgavesett, oppgaver og deloppgaver skal ha unike ID-er, lagres tellere. Oppgavesetttelleren lagres i filen «tellere.json» og inkrementeres hver gang et nytt oppgavesett lages. Videre befinner telleren for oppgave-ID seg i JSON-dataen for et oppgavesett. Oppgave-ID trenger bare å være unik per oppgavesett ettersom oppgaver lagres internt i et oppgavesett. Det samme gjelder deloppgaver som lagres internt i en oppgave, her lagres en ny teller for hver oppgave.

16.7.2 Oppgavesiden

Når tjeneren skal omdirigere en klient til oppgavesiden, gjøres dette med «render_template(«oppgaveside.html», ...). «render_template» bruker Jinja til å bygge den spesifiserte HTML-filen og vise inkludert data. (davidism. ThiefMaster., 2021, a). Oppgavesettene inkluderes som data i «render_template», og vises på brukerens skjerm ved en løkke som lager et nytt kort for hvert oppgavesett. Enda en løkke kjøres per oppgavesett for å vise oppgavekortene.



Figur 40: JSON-data-struktur. Et oppgavesett inneholder flere oppgaver med flere deloppgaver.

16.7.3 Innlasting oppgave

Når en bruker skal laste inn en oppgave, vil klienten sende en ≪POST≫-forespørsel til tjeneren som inkluderer oppgavesettets ID og oppgave-ID til den valgte oppgaven. Serveren omdirigerer da brukeren til ≪strukturbyggeren.html≫, og inkluderer ID-ene sendt fra klienten. Når klienten laster inn strukturbyggeren, kjøres automatisk ≪lastInn(oppgaveSettID, oppgaveID, oppgaveType, manuell)≫. Funksjonen henter oppgavedataen via en ≪GET≫-forespørsel. Tjeneren indekserer inn i minnet til riktig oppgave ved bruk av ID-ene. Klienten blir da tilsendt oppgaveinformasjon og alle tilhørende deloppgaver. Brukeren vil kunne navigere gjennom de forskjellige deloppgavene uten å måtte hente mer data fra tjeneren.

Dersom en bruker skulle ønske å laste inn startstrukturen på nytt, kan ≪lastInn(...)≫-funksjonen kjøres på nytt ved å trykke på ≪Startstruktur≫-knappen. En administrator vil også kunne laste inn fasitstrukturen.

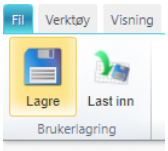
16.7.4 Lagring av oppgave

En administrator skal kunne lagre oppgavesett, oppgaver, deloppgaver og fasit. Dette gjøres ved at administratorbrukeren trykker på «Lagre»- eller «Lagre fasit»-knappene som sender en «POST»-forespørsel til tjeneren. Forespørselen inkluderer informasjonen om oppgaven som type, struktur, navn og beskrivelse. Dersom en versjon av oppgaven er lagret i tjenerens minne fra før, vil oppgavedataen bli oppdatert. Hvis ikke, lagres en ny oppgave i minnet, og blir skrevet til disk for permanent lagring. Den nye oppgaven registreres også i databasen via en databaseoperasjon.

16.7.5 Brukerdatalagring

Applikasjonen gir også brukere muligheten til å lagre sin struktur i en egen brukermappe. Dette kan være nyttig dersom en bruker jobber med en større oppgave og ønsker å lagre det vedkommende har gjort så langt. Brukerdata lastes ikke inn i minnet ved start, men hentes ut når en bruker sender en «GET»-forespørsel med «Last inn»-knappen.

For lagring av brukerdata sendes en «POST»-forespørsel til tjeneren som inneholder brukerdataen, og dataen blir lagret i brukermappen som hører til brukerens ID.



Figur 41: Knapper for å lagre brukerdata.

Del V

Evaluering

17 Utfordringer

17.1 JavaScript

Vi opplevde at JavaScript hadde en del svakheter når det gjaldt objektorientering. Språket mangler støtte for å kunne kopiere objekter, og om man bruker referanser til andre objekter som egenskaper til et annet objekt, vil det ikke være mulig å lagre dette objektet i JSON og laste det inn igjen. Dermed gikk vi over til et ID-system for objekter som skal lagres.

17.1.1 Beregning av start og sluttpunkt

Når kanter ble implementert, ville vi at selve kanten skulle starte og slutte i ytterkanten av nodene for å ikke være synlig i bakgrunnen av en gjennomsiktig node. I starten ble vanlig geometri brukt for å beregne disse punktene, men dette viste seg å være en relativt tungvint metode, og kantene ble lite nøyaktige. Senere testet vi vektormetoder for å løse de samme problemene, og dette ga mye bedre resultater. Vi fant ut at JavaScript ikke inkluderer funksjoner for vektorregning, dermed ble vektormetoder implementert i ≪vektormetoder≫-filen.

17.2 Norske tegn i JSON

Et problem vi støtte på under utviklingen, var at data omgjort til JSON-format ikke har koding for norske tegn som standard. Tidlig i prosjektet ble oppgavetittelen brukt som nøkkel for å lagre oppgavedata. Dersom en tittel inneholdt norske tegn, ville konverteringen til JSON-format endre de norske tegnene til diverse andre tegn. Dette skapte vanskeligheter med henting av oppgavedata, ettersom tjeneren ikke greide å finne oppgavene i minnet.

Et annet problem var at beskrivelser som inneholdt norske tegn, også ble feil etter lagring. Dette gjorde at beskrivelser ble vanskelige å lese ettersom de norske tegnene var byttet ut med andre tegn.

For å fikse problemet endret vi innlasting og lagring av JSON-data slik at UTF-8-koding ble brukt. Dette gjorde at de norske tegnene ble lagret og hentet ut på riktig måte. Vi endret også nøklene til å heller bruke ID-er som nøkler, noe som egentlig burde ha blitt gjort i utgangspunktet. Dette fikset også potensielle problemer relatert til oppgaver med samme tittel som kunne oppstått.

17.3 Node ID

I denne applikasjonen lagres noder med et heltall som ID. Når en ny node lages, vil en global tellervariabel brukes til å gi noden en ny unik ID. En utfordring vi fikk med dette systemet var at brukere kan bevege seg mellom deloppgaver. Dersom applikasjonen ikke endrer den globale tellervariabelen når brukeren bytter deloppgaver, kan nye noder plasseres med samme id. Dette var et problem vi hadde tidlig i prosjektet. Utfordringen besto hovedsaklig av å finne feilen. Feil ville bare oppstå av og til, slik at det var vanskelig å finne årsaken til problemet. Da feilen ble funnet, var problemet ganske enkelt å fikse, og det ble gjort ved å lagre en egen tellervariabel for hver deloppgave. Hver gang brukeren endret deloppgave ble den nye telleren lastet inn.

18 Diskusjon

Den første tiden i prosjektet ble brukt til å implementere grunnfunksjonaliteten i Strukturbyggeren. Siden lerretet er så viktig for applikasjonen, brukte vi mye tid på å utvikle et robust tegne- og visualiseringssystem for å gjøre det enklere å utvide applikasjonen uten å endre grunnfunksjonaliteten.

Da vi nærmet oss halvveis i prosjektet, var grunnfunksjonene til Strukturbyggeren og enkelte tjenerfunksjoner ferdig implementert. Det neste steget ble å implementere de forskjellige oppgavetypene og retting av disse.

Etter implementasjon av oppgaver og retting nærmet vi oss etterhvert den siste måneden i prosjektet. Her måtte innlogging via Feide, administratorrettigheter og lagring av brukerdata implementeres. Vi satte oss også som mål å implementere tilkobling til Canvas-plattformen dersom det skulle bli tid igjen til dette. Det ble ikke tid til å implementere denne funksjonaliteten, og dermed valgte vi å heller å rette og forbedre grunnfunksjonaliteten til applikasjonen.

Da vi begynte på prosjektet, bestemte vi oss for å redusere bruken av JavaScript-biblioteker og rammeverk for å unngå vedlikehold ved eventuelle endringer i disse. Dette innebar at implementasjonen ble gjort mer fra bunnen av, og krevde mer tid. Alternativet hadde vært å bruke biblioteker og rammeverk i større grad, som nok ville gjort utviklingen raskere, men ville som nevnt ført til økt vedlikeholdsbehov og redusert våre valgmuligheter ved utviklingen.

Del VI

Konklusjon

Strukturbyggeren oppfyller hovedmålene med oppgaven: Administrator kan enkelt legge til oppgaver for DAT200-faget og elever kan logge inn med Feide og løse oppgavene. I tillegg kan administrator se hvem som har gjort hvilke oppgaver, ved å bruke oversiktsiden. Applikasjonen er intuitiv og har en logisk oppbygning, samtidig som oppgavetypene er hensiktsmessige. Vi rakk ikke implementere Canvas-integrering, men dette trengs heller ikke for at applikasjonen skal være funksjonell.

19 Fremtidige utvidelser

Tidsfrister for oppgavesett oppgaver eller deloppgaver kan med fordel bli implementert, og gjerne med en kalender, så man kan se hva som skal leveres inn snart. Skulle dette bli implementert, må man nok utvide databasetabellene med ekstra kolonner for tidsfristen og i tillegg avvise innleveringer etter tidsfristen har gått ut.

Enhetstester ville forenklet videre utvikling og begrenset nødvendig testing, men slike tester finnes ikke per dags dato i JavaScript, så man måtte ha importert et eget bibliotek. Enhetstester hadde også passet bedre med en enda større grad av objektorientering, og da hadde kanskje det vært lurt å gå over til utvidelser av JS som tilbyr dette, eller et annet språk og rammeverk.

Underveis har vi testet bruk av mobil og har funnet ut at det ikke er så mye som står i veien for å gå videre med utviklingen for mobil, men at mobiltelefon som grensesnitt ikke er spesielt egnet for applikasjonen på grunn av størrelsen og unøyaktigheten ved å bruke fingrene. Strukturbyggeren er mye mer egnet for nettbrett og penner, men så er det heller ikke så mange som har nettbrett lenger. Kanskje kan utvikling for mobiltelefon bli viktigere etterhvert som telefoner som kan brettes ut til større skjermer, gjerne med penn, blir mer utbredt. I så fall er den største endringen som kreves å gå over fra «muse-hendelser» til «berørings-hendelser» og «penne-hendelser».

Penneverktøy kunne vært nyttig for å kunne drodle og forklare.

Sandkasse-modus kan bli implementert for å gi tilgang til tegneprogrammet uten innlogging. Modusen kunne også gitt tilgang til alle verktøy slik at en bruker kunne brukt verktøyene til å leke med tre- og grafstrukturer.

Del VII

Referanser

- Computerphile. (2017. 4. januar). Dijkstra's Algorithm - Computerphile [Video]. YouTube.

  https://youtu.be/GazC3A4OQTE

- Countryman, M. (2019, desember). Flask-Login. Readthedocs.

  https://flask-login.readthedocs.io/en/latest/

- davidism. ThiefMaster. (2021, 4. mai, a). Templates. Palletsprojects.

  https://flask.palletsprojects.com/en/1.1.x/tutorial/templates/

- davidism. ThiefMaster. (2021, 9. mai, b). Jinja. Palletsprojects.

  https://jinja.palletsprojects.com/en/2.11.x/

- Ecma international. (2017, desember). ECMA-404. Ecma-international.

  https://www.ecma-international.org/publications-and-standards/standards/ecma-404/

- Yang, H. (2021, mars). Authlib: Python Authentication. Authlib.

  https://docs.authlib.org/en/latest/basic/intro.html

- OktaDev. (2019. 5. november). An Illustrated Guide to OAuth and OpenID Connect [Video]. YouTube.

  https://www.youtube.com/watch?v=t18YB3xDfXI

- Uninett. (u.å.). Feide. Hentet 9. mai 2021 fra

  https://docs.feide.no/general/feide_overview.html

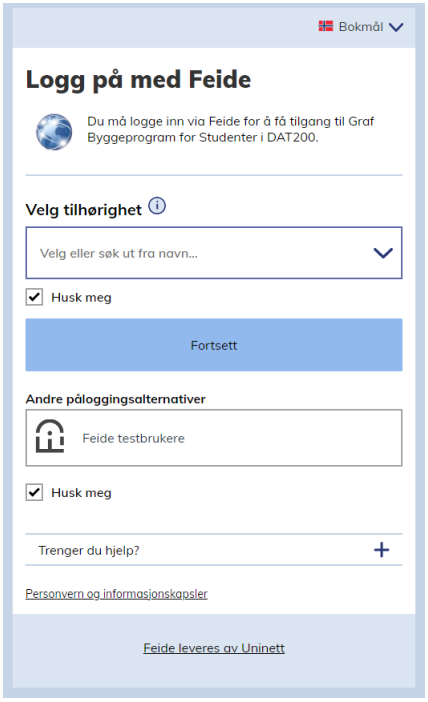
Del VIII

Brukerveiledning

20 Innlogging.

1. Innlogging:

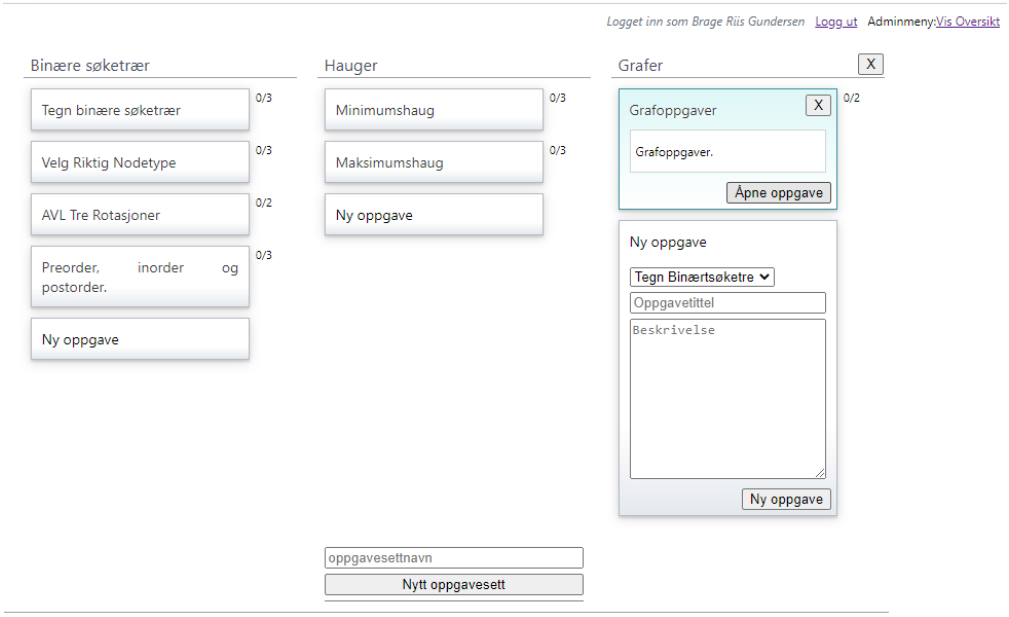
   (a) Logg inn med Feide for å få tilgang til applikasjonen.



Figur 42: Innlogging via Feide kreves for å ta i bruk applikasjonen.

21 Oppgavesiden

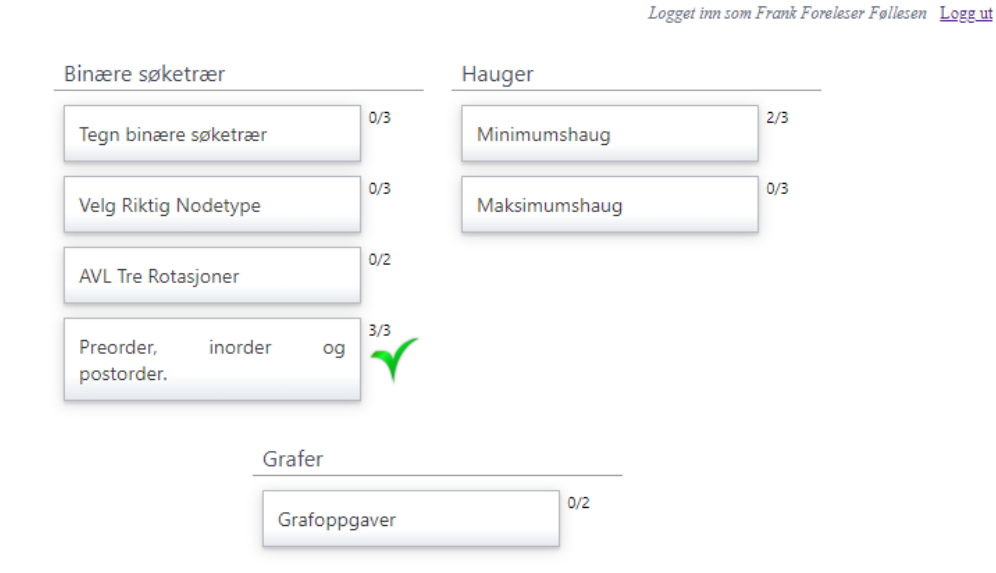
21.1 Administrator



Figur 43: Oppgavesiden for en administrator.

1. Lage nytt oppgavesett:
   (a) Skriv inn oppgavesettnavn i datafeltet på skjermen.
   (b) Trykk på «Nytt oppgavesett»-knappen.
2. Lage en ny oppgave:
   (a) Trykk på «Ny oppgave»-feltet som befinner seg under oppgavesettet oppgaven skal tilhøre.
   (b) Velg oppgavetype fra menyen og skriv inn oppgavetittel og beskrivelse.
   (c) Trykk på «Ny oppgave»-knappen.
3. Gå til en oppgave:
   (a) Trykk på oppgaven.
   (b) Trykk på «Åpne oppgave»-knappen.
4. Se oversikt over brukere som har løst de ulike oppgavene:
   (a) Trykk på «Vis Oversikt»-lenken som befinner seg øverst i høyre hjørnet på skjermen.

Figur 44: Oversiktsiden. Er kun tilgjengelig for administratorer.



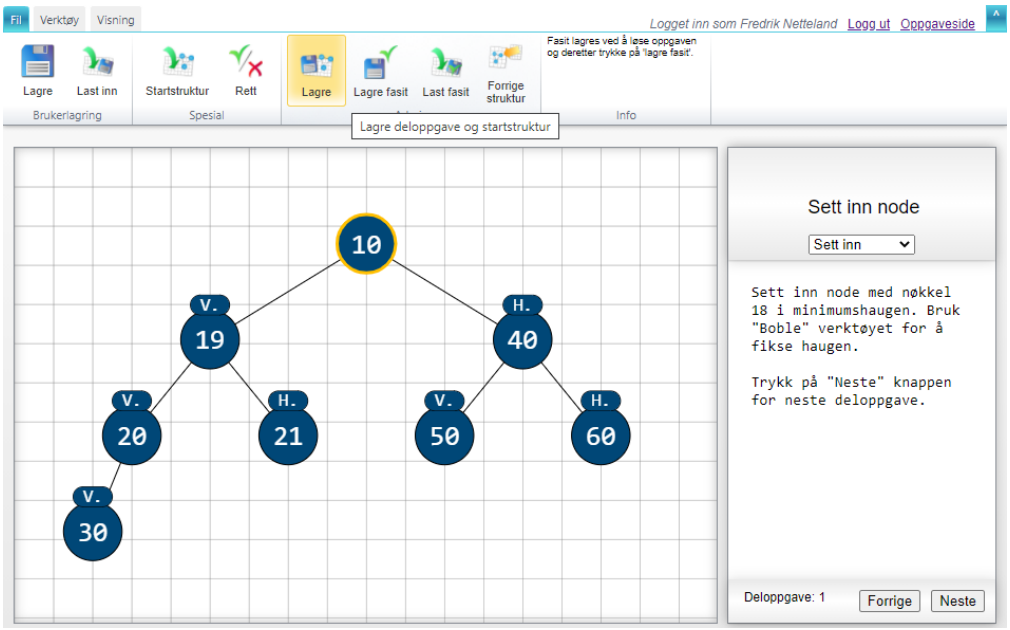
Figur 45: Oppgavesiden for en bruker.

21.2 Bruker

1. Gå til en oppgave:
   (a) Trykk på oppgaven.
   (b) Trykk på «Åpne oppgave»-knappen.
2. Sjekk hvor mange deloppgaver som er løst:
   (a) Antall løste deloppgaver vises til høyre for oppgaven.

22 Oppgaver

22.1 Administrator



Figur 46: Strukturbyggeren sett fra en administrator.

1. Lag ny eller oppdater deloppgave:
   (a) Naviger til riktig deloppgave med «Neste» og «Forrige»-knappene.
   (b) Trykk på tittelen og skriv inn.
   (c) Trykk på oppgavebeskrivelsen og skriv inn oppgavebeskrivelse og oppgaveinfo.
   (d) Velg deloppgavetype fra menyen. Merk: denne menyen vises bare dersom det finnes flere deloppgavetyper.

(e) Dersom oppgaven skal ha en startstruktur, tegn startstrukturen inn på lerretet.

i. Dersom oppgaven skal ha samme startstruktur som den forrige deloppgaven, kan «Forrige struktur»-knappen trykkes på for å laste inn startstruktur fra forrige deloppgave.

(f) Trykk på «Lagre»-knappen som befinner seg i filmenyen.

2. Lagre fasit:

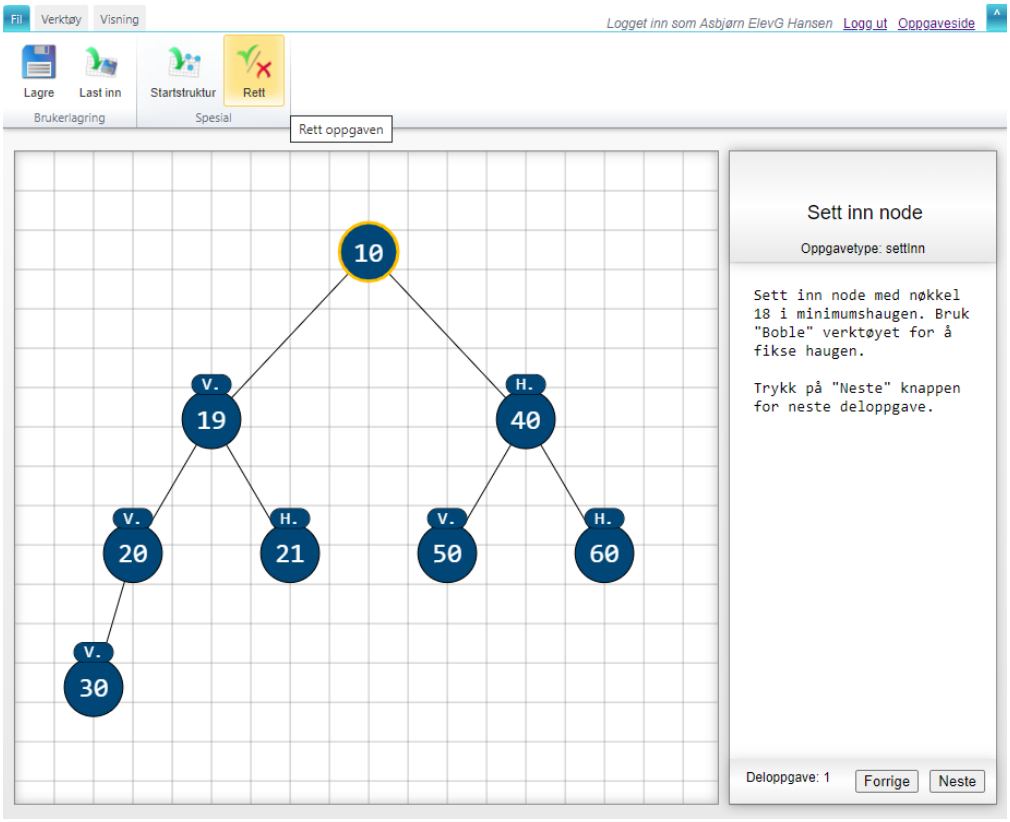
(a) Tegn riktig struktur på lerretet.

(b) Trykk på «Lagre fasit»-knappen. Merk: dersom denne knappen ikke vises betyr det at fasit lages automatisk etter at man har trykket «Lagre».

3. Last inn fasit.

(a) Trykk på «Last fasit»-knappen.

22.2 Bruker



Figur 47: Strukturbyggeren sett fra en bruker.

1. Løse en oppgave:

(a) Les av oppgavetype og oppgavebeskrivelse for å forstå oppgaven.

(b) Bruk de vedlagte verktøyene til å løse oppgaven beskrevet i oppgavebeskrivelsen.

(c) Trykk på ≪Rett≫-knappen for å sjekke om svaret er riktig.

(d) Trykk på ≪Neste≫-knappen for å gå til neste deloppgave. Merk: denne knappen vises bare dersom det er flere deloppgaver.

2. Resett lerretinnhold:

   (a) For å resette lerretinholdet, trykk på ≪Startstruktur≫-knappen.

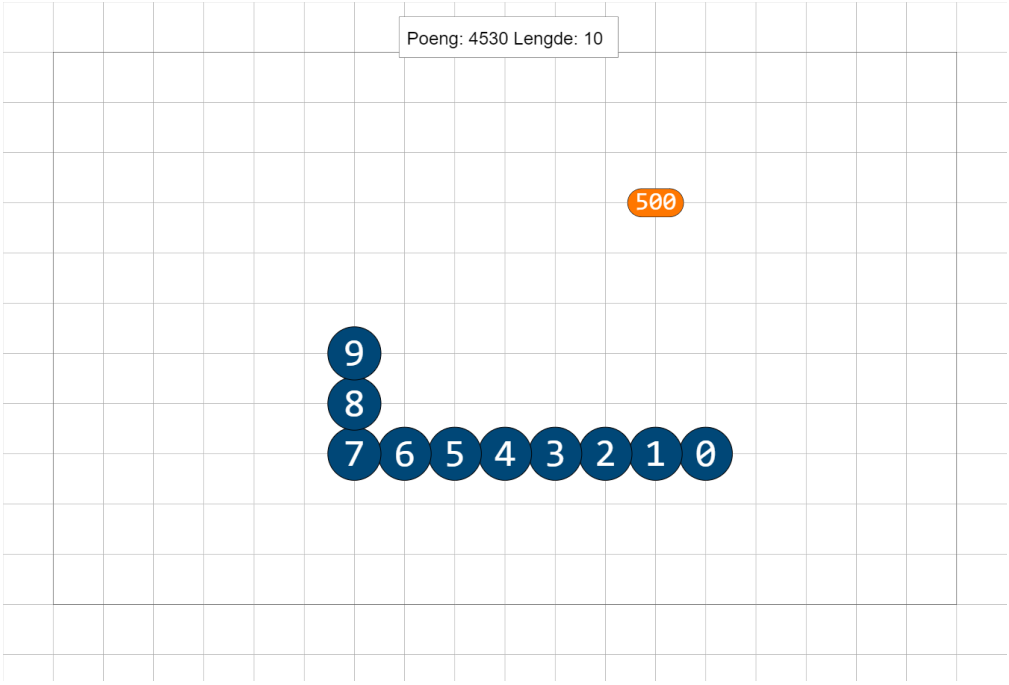
3. Brukerlagring:

   (a) For å lagre framgang på en oppgave, trykk på ≪Lagre≫-knappen.

   (b) For å laste inn lagret framgang på en oppgave, trykk på ≪Last inn≫-knappen.

22.3 Ekstra

Det er inkludert to spill som man kan starte fra nettleserkonsollen med «snake()» og «pong()», som er laget med funksjonalitet som allerede finnes i Strukturbyggeren. Det er lurt å starte spillene fra et tomt lerret.



Figur 48: Snake

Figur 49: Pong