Norges teknisk–naturvitenskapelige universitet
Institutt for datateknologi og informatikk

TDT4102 Prosedyre- og objektorientert programmering Vår 2022

Øving 10

Frist: 2022-04-01

Aktuelle temaer for denne øvingen:

- Klasser, arv og GUI
- Rekursjon

Generelle krav:

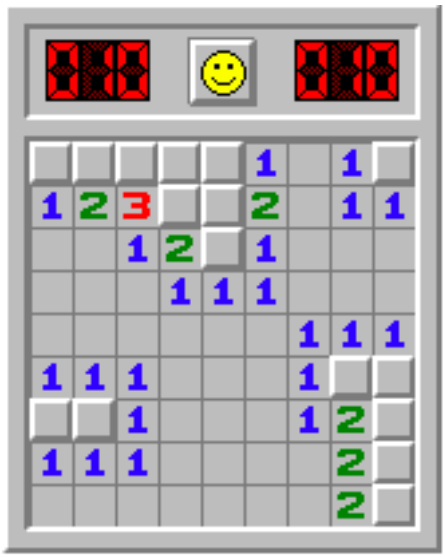
- Bruk de eksakte navn og spesifikasjoner gitt i oppgaven.
- Teorioppgaver besvares med kommentarer i kildekoden slik at læringsassistenten enkelt finner svaret ved godkjenning.
- 70% av øvingen må godkjennes for at den skal vurderes som bestått.
- Øvingen skal godkjennes av stud.ass. på sal.
- Det anbefales å benytte en programmeringsomgivelse(IDE) slik som Visual Studio Code.

Anbefalt lesestoff:

- FLTK-widgets og Minesveiper

Bakgrunn for oppgavene

Tema for denne øvingen er spillet Minesveiper. Spillbrettet består av et rektangulært rutenett av firkanter. Under et gitt antall tilfeldige ruter ligger det miner, og spilleren må åpne alle rutene som ikke inneholder miner for å vinne spillet. Hvis spilleren åpner en rute som inneholder en mine, har han tapt. Ruter som ikke inneholder en mine har et nummer fra null til åtte som representerer hvor mange miner det finnes i naborutene. Ved hjelp av dette er det som oftest mulig å bruke logikk for å avgjøre hvor minene ligger. For å hjelpe spilleren med å huske hvilke ruter som er miner, tilbyr spillet at miner kan flagges. Dette gjøres ved å høyreklikke på en mine, da er den markert og kan ikke åpnes før flagget fjernes. Spillet er også laget slik at dersom en rute åpnes, og det ikke er noen miner rundt, åpnes alle rutene rundt. Et bilde av spillet halvveis kan se slik ut:



Et halvspilt minesveiperspill.

Hvis spillereglene ikke kom tydelig fram av denne teksten, kan man lese mer fra lenken som er lagt inn under anbefalt lesestoff eller prøve spillet selv her: MinesveiperOnline.

1 Last ned de utdelte filene, og opprett et nytt prosjekt (0%)

Last ned de utdelte filene via TDT4102-extensionen på samme måte som i tidligere øvinger. Hent også inn «Configuration Only»-templaten slik at programmet kan bygges.

Når du kjører programmet bør du nå få opp et spillbrett med 10 × 10 ruter. Når du klikker på rutene skjer det ikke noe annet enn at man ser at de blir trykket på. I filen `main.cpp` ser du at bredde og høyde for vinduet er satt til 10. Dette representerer antall ruter. I tillegg er antall miner satt til 3. Deretter opprettes et `MinesweeperWindow`, før `gui_main()` kalles og gir kontroll til grafikkvinduet.

2 Klassen Tile (25%)

Når man skal programmere, enten det er et spill eller noe annet er det viktig å velge riktig datastruktur. Siden spillet består av et rutenett, kan det representeres med en `vector` som holder `shared_ptr` til alle rutene. For hver rute må spillet holde styr på om ruten har blitt åpnet eller ikke, er flagget eller ikke og om den inneholder en mine eller ikke. I tillegg må ruten endres når den trykkes på. Her skal vi bruke en klasse `Tile` som arver fra `Fl_Button` til å representere hver rute. `Fl_Button` bør være kjent fra tidligere øvinger.

Klassen finner du deklarasjonen av i `Tile.h` og implementasjonen ligger i `Tile.cpp`. Som du ser av den utdelte koden gjenbrukes `Fl_Button`-klassens konstruktør. I tillegg er det lagt til 2 medlemsfunksjoner, `void set_label(string)` og `void set_label_color(Color)`, som setter teksten som skal stå på ruten og fargen til denne. Det er også lagt til medlemsfunksjoner, `void open()` og `void flag()`, for å åpne og flagge en rute. Det du ser av kode i `void open()`-funksjonen er det som gjør at man ser at rutene på brettet er trykket på (dette vil komme fram når du implementerer `MinesweeperWindow` i oppgave 3). Hver rute har også en state av typen `Cell`, med tilhørende get-funksjon, som er en enumtype som viser om ruten er lukket, åpnet eller flagget. I kildekoden er det også definert noen maps som kan komme til nytte.

a) I minesveiper er det viktig å ha kontroll på om en rute er en mine eller ikke. Legg derfor til medlemsvariabelen `bool isMine`, den skal være privat og ha defaultverdien `false`. Lag også get- og set-funksjoner for `isMine`.

b) Nå kan vi oppdatere `open()`-funksjonen. Den skal nå gjøre følgende:

- Dersom ruten er flagget eller åpen skal ingenting skje.
- Dersom ruten er lukket skal den settes som trykket på (kodelinjen som er der fra før) og **state** skal settes til open. Dersom ruten også er en mine skal vi sette labelen til en rød X.

c) Videre kan vi oppdatere `flag()`-funksjonen som foreløpig ikke gjør noe. Den skal nå gjøre følgende:

- Dersom ruten er flagget fra før skal `state` settes til closed og symbolet for at den er flagget skal fjernes.
- Dersom ruten ikke er flagget fra før skal `state` settes til flagget, labelen skal settes til symbolet for flagg og fargen på labelen settes til blå.

Hint: Benytt map'et `CellToSymbol` som ligger lenger opp i filen.

d) Vi skal nå lage enda en medlemsfunksjon:

- `void setAdjMines(int n)`

Denne skal ta inn et heltall mellom 1 og 8 (vi ser bort ifra 0 foreløpig) som symboliserer hvor mange miner ruten har rundt seg. Basert på dette skal labelen settes til antall miner rundt ruten. Labelen skal også ha en unik farge for hvert tall.
Hint: Benytt map'et `minesToColor` som ligger øverst i `Tile.cpp` opp i filen.

3 Klassen MinesweeperWindow (35%)

Nå som hver rute har en passende datastruktur kan vi bevege oss videre til spillvinduet. I filen `MinesweeperWindow.h` finner du klassen `MinesweeperWindow` som arver fra `AnimationWindow`-klassen, som bør være kjent fra tidligere øvinger. Som du ser i konstruktøren til klassen, i `MinesweeperWindow.cpp`, gjenbrukes `AnimationWindow`-klassens konstruktør. I tillegg opprettes det riktig antall `Tile`-objekter som fordeles utover vinduet og legges inn i medlemsvariabelen `tiles`, som er en `vector<shared_ptr<Tile>`. I tillegg inneholder klassen en del hjelpefunksjoner som kan bli nyttige i oppgavene som følger. Vi skal nå oppdatere denne klassen videre til et fungerende spill.

a) Når spillet starter skal det plasseres et bestemt antall miner på brettet. Antallet er gitt i argumentet `mines` til konstruktøren. Som du ser setter konstruktøren nå alle rutene til å ikke ha miner (ettersom dette er defaultverdien til en `Tile`). Du skal nå plassere `mines` antall miner på brettet. Gjør dette i konstruktøren. Pass på å at du ikke plassere flere miner i samme rute.

Callback-funksjonen `cb_click()` sørger for at det ved et klikk fra spilleren kalles enten `flagTile(Point xy)` eller `openTile(Point xy)`. Disse er foreløpig tomme, men du skal nå implementere logikk i disse funksjonene. Som du ser er input-parameteren til begge funksjonene et punkt på vinduet. Dette er det punktet spilleren trykket på.

b) I funksjonen `flagTile(Point xy)` skal du ut ifra punktet som er trykket på finne riktig tile. Deretter skal denne flagges.

Hint: Sjekk ut medlemsfunksjonen `shared_ptr<Tile>& at(Point xy)`.

c) I funksjonen `openTile(Point xy)` skal du ut ifra punktet som er trykket på finne riktig tile. Deretter skal denne åpnes.

d) I begge funksjonene vil vi bare at det skal skje noe med ruten dersom den er i en bestemt `state`. For `openTile(Point xy)` skal det bare skje noe hvis ruten er lukket. For `flagTile(Point xy)` skal det skje noe hvis ruten er enten flagget eller lukket. Implementer logikk i begge funksjonene slik at dette er tilfellet.

Dette er et fint tidspunkt å teste programmet ditt på. Sjekk om du kan flagge og åpne ruter i vinduet ditt. Husk at du kan justere størrelse på brettet og antall miner i `main()`-funksjonen.

Før vi implementerer mer logikk for `openTile(Point xy)` trenger vi en funksjon som kan avgjøre hvor mange miner det er rundt den ruten vi åpner.

e) Implementer funksjonen `int countMines(vector<Point> points)`. Denne tar inn en vektor med punkter og skal telle hvor mange av rutene, som hører til disse punktene, som er miner. Deretter skal dette antallet returneres. Du trenger ikke å ta høyde for at punktene kan være utenfor vinduet.

f) Nå kan vi oppdatere `openTile(Point xy)`.Denne skal nå etter å ha åpnet ruten sjekke om den er en mine. Dersom den ikke er det skal funksjonen sjekke hvor mange miner som ligger rundt. Hvis dette er flere enn 0 skal ruten markeres med dette tallet og en unik farge.

Hint: Medlemsfunksjonen `vector<Point> adjacentPoints(Point xy)` returnerer en vektor med de punktene som hører til naborutene til ruten representert av inputpunktet. Den sjekker også at punktene faktisk er på brettet og returnerer kun disse.

Dersom ruten ikke har noen miner rundt seg er det åpenbart at spilleren kan klikke på alle rutene rundt uten at det er noen fare for å trykke på en mine. Dette kan likevel være kjedelig. Derfor er Minesveiper laget slik at dersom en rute åpnes, og ikke har noen miner rundt seg, skal alle rutene rundt åpnes rekursivt.

g) Oppdater `openTile(Point xy)`. Denne skal nå, dersom det ikke er noen miner rundt, åpne alle naborutene. Videre skal disse naborutene åpne alle sine naboruter dersom de

heller ikke har noen miner rundt seg osv. Dersom naborutene har miner rundt seg skal de markeres med dette tallet som vanlig.

På dette tidspunktet er det naturlig å teste at logikken fungerer som den skal. Prøv deg fram med å trykke på brettet du nå får opp.

4 Spill-logikk (30%)

Et spill er ikke veldig spennende dersom det ikke er mulig å vinne eller tape. Du skal nå derfor implementere logikk for å avgjøre om spillet er vunnet eller tapt. Et tips er at spillet tapes med en gang dersom spilleren trykker på en mine. For å avgjøre om spillet er vunnet må man derimot holde styr på hvor mange ruter som er åpnet, når alle ruter som ikke er mine er åpnet, uten at spillet er tapt, er spillet vunnet.

a) Legg til logikk for å avgjøre om spillet er tapt.

b) Legg til logikk for å avgjøre om spillet er vunnet.

c) Gi spilleren tilbakemelding om spillet er vunnet eller tapt i form av en `Fl_Output` som dukker opp i grafikkvinduet. Denne widgeten kan vise tekst og du velger selv hvor teksten skal plasseres og hvordan den skal se ut.
Hint: `Fl_Output` er en FLTK-widget (Se her for dokumentasjon). Denne kan legges til som en medlemsvariabel og initialiseres i konstruktøren til `MinesweeperWindow`.

5 Ekstra funksjonalitet (10%)

En god minesveiperspiller vil flagge alle ruter han tror er miner. Det er en fordel for spilleren å vite hvor mange miner det er igjen hvis alle flaggede ruter er miner.

a) Gi spilleren tilbakemelding i vinduet på hvor mange miner som gjenstår hvis man antar at de flaggede rutene er miner. Du kan selv velge om du skal gi denne tilbakemeldingen oppå rutene, eller utvide vinduet og gi tilbakemeldingen under rutene. Du kan endre størrelsen på vinduet i `MinesweeperWindow`-konstruktøren. Når `AnimationWindow`-konstruktøren kalles bestemmes størrelsen på vinduet. Hvilket argument må du endre på for at vinduet skal blir høyere? Merk at `height` og `width` er vinduets høyde og bredde i antall ruter.

Det er alltid surt å tape et spill, men det kan være greit å få tilbakemelding på hva man gjorde rett og galt. I minesveiper gis denne tilbakemeldingen i form av at alle rutene som var miner markeres med en mine (i denne øvingen et rødt kryss) ved spillslutt dersom man har tapt. Ved seier markeres alle rutene som var miner med et flagg.

b) Marker alle ruter som var miner med en rød X ved spillslutt dersom spillet er tapt. Flagg alle ruter som var miner ved spillslutt dersom spillet er vunnet.

I tillegg er det greit å kunne starte spillet på nytt uten å måtte lukke og åpne vinduet mellom hver gang.

c) Oppdater programmet slik at det ved spillslutt ikke bare dukker opp en tekst. Det skal i tillegg dukke opp to knapper **quit** og **restart**. Ved å trykke på **restart** skal minene fordeles tilfeldig på nytt og alle ruter lukkes slik at spilleren kan spille på nytt, funksjonen under kan brukes for å «nullstile» ruter. Ved trykke på **quit** skal vinduet lukkes.

```
this->value(0);
```