



Norges teknisk-naturvitenskapelige
universitet
Institutt for datateknologi og
informatikk

TDT4102 Prosedyre-
og objektorientert
programmering
Vår 2019

Øving 10

Frist: 2019-03-22

Aktuelle temaer for denne øvingen:

- Klasser, arv og GUI

Generelle krav og anbefalinger:

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

Anbefalt lesestoff:

- Kapittel 16 og **Minesveiper**

Bakgrunn for oppgavene

Tema for denne øvingen er spillet Minesveiper. Spillbrettet består av et rektangulært rutenett av firkanter. Under et satt 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 linken som er lagt inn under anbefalt lesestoff eller prøve spillet selv her: [MinesveiperOnline](#).

Siden det å spille minesveiper i konsollvinduet fort blir tungvint, skal vi bruke et grafikkprosjekt i denne øvingen. For å komme i gang med øvingen har vi laget et kodeskjelett som oppretter et brett med ruter.

1 Sette opp et grafikkprosjekt og laste inn utdelte filer.(0%)

Start et nytt grafikkprosjekt.

Last inn de fem utdelte filene `main.cpp`, `Tile.h`, `Tile.cpp`, `MinesweeperWindow.h` og `MinesweeperWindow.cpp`. 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 er 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 Datastrukturen 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_ref** som holder 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 **struct Tile** som arver fra klassen **Button** til å representere hver rute. **Button**-klassen bør være kjent fra tidligere øvinger, hvis ikke kan kapittel 16.4.2 i læreboka være til hjelp.

Datastrukturen finner du definisjonen av i **Tile.h** og implementasjonen ligger i **Tile.cpp**. Som du ser av den utdelte koden gjenbrukes **Button**-klassens konstruktør. I tillegg er det lagt til 2 medlemsfunksjoner som setter teksten som skal stå på ruten og fargen til denne. Det er også lagt til medlemsfunksjoner 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 vi implementerer **MinesweeperWindow** i oppgave 3). Hver rute har også en state, dette 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**. Denne verdien skal settes når en rute opprettes og må derfor være inputparameter til konstruktøren, her skal den settes med defaultverdi **false** og må derfor stå sist i parameterlisten. Dette gjør at vi kan opprette en **Tile** uten å spesifisere om den er mine.
- b) Nå kan vi oppdatere **open()**-funksjonen. Den skal nå gjøre følgende:
 - Dersom ruten er flagget skal ingenting skje.
 - Dersom ruten ikke er flagget 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 implementere 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 lenger 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.cpp** finner du klassen **MinesweeperWindow** arver fra **Window**-klassen i **Graph_lib**, som bør være kjent fra tidligere øvinger. Hvis ikke kan kapittel 12.6 og 16.3 være til hjelp. Som du ser i konstruktøren til klassen gjenbrukes **Window**-klassens konstruktør. I tillegg opprettes det riktig antall **Tile**-objekter som fordeles utover vinduet og legges inn i

medlemsvariabelen `tiles`, som er en `Vector_ref` datatype. 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). Dette skal du endre på ved å plassere ut riktig antall miner på tilfeldige ruter på brettet. Gjør dette i konstruktøren.

Merk: Pass på at du ikke plasserer miner i samme rute flere ganger.

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 trykker 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 `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. Dersom det virker som om rutene ikke oppdaterer seg til riktige tidspunkt, men heller etter nye trykk, kan det være at de ikke tegnes på nytt når du vil. F. eks. etter

du har endret label. To funksjoner som kan hjelpe med dette og blir nyttige i de neste oppgavene er `redraw()` og `flush()`.

Nyttig å vite: `redraw()` og `flush()`

`redraw()` er en medlemsfunksjon som alle `Widget` har. Når funksjonen kalles på en `Widget` vil hele det rektangulære området den spenner over bli tegnet på nytt ved neste anledning. Det betyr at du ikke nødvendigvis umiddelbart vil se oppdateringen. Noen endringer på et element vil ikke automatisk oppdatere det som vises i vinduet, som f.eks. endring av farge eller label-tekst. Når disse små endringene skal gjenspeiles i vinduet må du kalle `redraw()` på de elementene det gjelder.

`flush()` er en medlemsfunksjon som `Window` har. Denne medlemsfunksjonen tvinger fram tegning av elementer som tidligere har blitt markert med `redraw()`.

`Window` er også en `Widget`, det gjør at medlemsfunksjonen `redraw()` til et vindu vil klargjøre alle elementer i vinduet til å tegnes ved neste anledning.

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 feedback på om spillet er vunnet eller tapt i form av en tekst som dukker opp i grafikkvinduet.

5 Ekstra funksjonalitet (10%)

En god minesveiperspiller vil ønske å flagge alle ruter han vet at er miner. Derfor er det greit å vite hvor mange miner som er igjen utifra de som er flagget. Dette kan både være en fordel for å få oversikt og å avgjøre hvilke ruter som er miner og ikke når det er få ruter igjen.

- a) Gi spilleren feedback i vinduet på hvor mange miner som gjenstår på de rutene som ikke er flagget, dersom de som er flagget er miner.

Det er alltid surt å tape et spill, men det kan være greit å få feedback på hva man gjorde rett og galt. I minesveiper gis denne feedbacken 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. Flagget 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. Ved trykk på **quit** skal vinduet lukkes.