Norges teknisk–naturvitenskapelige universitet
Institutt for datateknikk og informasjonsvitenskap

TDT4102 Prosedyre- og objektorientert programmering Vår 2017

Øving 9

Frist: 2017-03-17

Mål for denne øvinga:

- Selvstending utvikling av klasser
- Repetisjon av dynamisk minnehåndtering
- Installasjon og bruk av tredjeparts bibliotek

Generelle krav:

- Følg spesifikasjonene og bruk de klasse- og funksjonsnavn som er gitt i oppgaven.
- Det er valgfritt om du vil bruke en IDE (Visual Studio, XCode), men koden må være enkel å lese, kompilere og kjøre.

Anbefalt lesestoff:

- Kapitler 6 og 10, Absolute C++ (Walter Savitch)
- http://www.sfml-dev.org/tutorials/2.3/
- Veiledninger til bruk av SFML med Visual Studio og Xcode på It's learning

Bakgrunn for oppgavene

I denne øvingen skal du lage 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. Siden spilleren skal ha mulighet til selv å stille vanskelighetsgraden til spillet, må arrayet eller arrayene som representerer spillebrettet være dynamiske.

Siden det å spille minesveiper i konsollvinduet fort blir tungvint, skal vi installere et bibliotek kalt SFML som lar oss lage et grafisk brukergrensesnitt. For å hjelpe deg med å komme igang med øvingen og SFML, har vi laget kode som bruker dette biblioteket og som definerer rammene for oppgaven. Denne koden kan lastes ned fra It's learning (`oving09.zip`), og består av filene `Minesweeper.cpp` og `Minesweeper.h` som inneholder et skall for klassen `Minesweeper`, og en `main.cpp` som benytter SFML og klassen `Minesweeper`.

For at koden i `main.cpp` skal virke, er det viktig at du implementerer `Minesweeper`-klassens `public`-metoder nøyaktig som skissert i oppgavene. Utover det står du egentlig ganske fritt til å implementere spillet som du vil, men oppgavene vil i noen tilfeller dirigere deg mot en mulig løsning.

1 Sette opp prosjektet og installere SFML (0%)

Last ned og sett opp din IDE til å bruke SFML. På It's learning finner du guider som leder deg gjennom prosessen steg for steg for Visual Studio og Xcode. Bruk disse, og spør en stud.ass. eller und.ass. på sal hvis du sitter fast. Hvis du ikke bruker Visual Studio eller Xcode kan du sjekke ut http://www.sfml-dev.org/tutorials/2.3/ for mer informasjon om SFML på din platform.

Lag et nytt prosjekt og legg til de vedlagte filene. (Brukere av Visual Studio og Xcode har allerede gjort dette.) Pakk ut `oving9-vedlegg.zip` og legg til `Minesweeper.h`, `Minesweeper.cpp` og `main.cpp` i prosjektet ditt. Pass også på at filen `sansation.ttf` ligger i prosjektmappa slik at programmet kan finne den når du kjører det. Du trenger også disse pakkene fra SFML for å kjøre den utdelte koden:

- `sfml-system`
- `sfml-window`
- `sfml-graphics`

2 Datastruktur (35%)

a) 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, vil det være naturlig å bruke en tabell for å representere tilstanden til spillebrettet. For hver rute må spillet holde styr på om ruten har blitt åpnet eller ikke og om den inneholder en mine eller ikke. Her skal vi bruke en `struct` til å representere dette (legg til denne i `minesweeper.h`):

```
struct Tile {
    bool open;
    bool mine;
};
```

Når brukeren starter en ny runde, vil det grafiske brukergrensesnittet opprette et `Minesweeper`-objekt. Konstruktøren mottar argumenter som spesifiserer hvor stort brettet skal være og hvor mange miner det skal inneholde. Brukergrensesnittet vil kalle på funksjonene `isTileOpen()`, `isTileMine()` og `numAdjacentMines()` hver gang det skal tegne brettet, og det vil kalle på `openTile()` når spilleren klikker på en rute.

b) Implementer konstruktøren og destruktøren til `Minesweeper`. Sørg for at alle rutene er lukket ved starten av spillet. Du kan selv velge hvordan du vil implementere tabellen. Hva må tabellen inneholde?

Hint: Du kan bruke både endimensjonal og todimensjonal tabell, i tillegg til å velge mellom dynamisk allokerte arrays eller bruk av `std::vector`.

c) Implementer følgende funksjoner:

- `Minesweeper::isTileOpen()` skal returnere om en rute har blitt åpnet eller ikke.
- `Minesweeper::isTileMine()` skal returnere om en rute inneholder en mine eller ikke.
- `Minesweeper::openTile()` skal markere en rute som åpnet.

3 Spill-logikk (35%)

Neste steg er å legge til miner og håndtere disse på riktig måte.

- Når spillet starter skal det plasseres et bestemt antall miner på brettet. Antallet er gitt i argumentet `mines` til konstruktøren.

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

- Hvis spilleren åpner en rute som inneholder en mine, er spillet over og `isGameOver()` skal returnere `true`.
- `numAdjacentMines()` skal returnere hvor mange av naborutene som inneholder miner.
  Naboskap inkluderer diagonale naboer. Hver rute har altså opptil 8 naboer. Pass på at dette også blir riktig for ruter langs kanten av brettet.

4 Ekstra funksjonalitet (30%)

a) (15%) Endre `openTile()` slik at hvis ruten ikke inneholder en mine og ingen av naborutene inneholder miner, åpnes også alle naborutene. Gjenta dette rekursivt for naborutene slik at store områder uten miner kan åpnes automatisk.

Hint: Hvis du ikke er kjent med rekursjon kan det være nyttig å lese kapittel 13 i pensumboka.

b) Legg til logikk for å avgjøre om spillet er vunnet eller tapt. Gi tilbakemelding til brukeren om resultatet når spillet er over. Valgfritt: Gjør dette i det grafiske vinduet, ikke i konsollen.

c) Valgfritt: Modifiser den utdelte koden og din egen kode som nødvendig for å la brukeren markere miner med flagg ved å høyreklikke på en rute. Hvis man høyreklikker igjen på flagget, skal flagget fjernes.

Lag logikk for å vise brukeren hvor mange miner som gjenstår.