Del III: Level Editor

Del 3 av eksamen er programmeringsoppgaver. Denne delen inneholder **13 oppgaver** som til sammen gir en maksimal poengsum på ca. 180 poeng og teller ca. **60 %** av eksamen. Hver oppgave gir *maksimal poengsum* på **5 - 20 poeng** avhengig av vanskelighetsgrad og arbeidsmengde.

Den utdelte koden inneholder kompilerbare (og kjørbare) *.cpp*- og *.h*-filer med forhåndskodede deler og en full beskrivelse av oppgavene i del 3 som en PDF. Etter å ha lastet ned koden står du fritt til å bruke et utviklingsmiljø (VS Code) for å jobbe med oppgavene.

På Inspera kan du laste ned *.zip*-filen om det **IKKE** fungerer å hente ut utdelt kode via fagets utvidelse (*extension*), og senere laste opp den nye *.zip*-filen med din egen kode inkludert.

Sjekkliste ved tekniske problemer

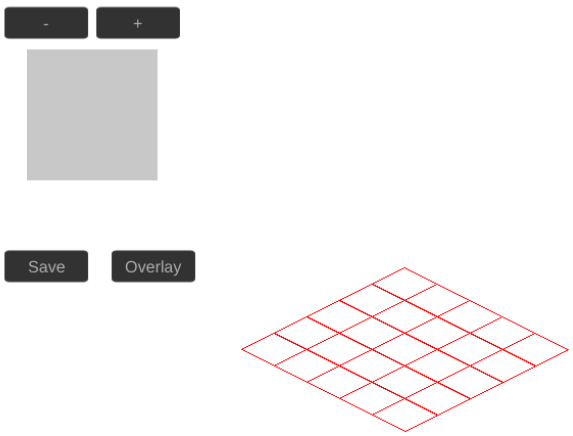
Hvis prosjektet du oppretter med den utdelte koden ikke kompilerer (før du har lagt til egne endringer) bør du rekke opp hånden og be om hjelp. Mens du venter kan du prøve følgende:

1. Sjekk at prosjektmappen er lagret i mappen `C:\temp`.
2. Sjekk at navnet på mappen **IKKE** inneholder norske bokstaver (*Æ, Ø, Å*) eller mellomrom. Dette kan skape problemer når du prøver å kjøre koden i VS Code.
3. Om det **IKKE** fungerer å hente ut utdelt kode via fagets utvidelse kan du prøve metoden som går ut på å laste ned zip-filen med utdelt kode fra Inspera:
   - Last ned .zip-filen fra Inspera og pakk den ut (unzip). Lagre filen i `C:\temp` som **eksamenTDT4102_kandidatnummer**. Du skal altså skrive inn *ditt eget kandidatnummer* bak understreken.
   - Åpne mappen i VS Code. Bruk deretter følgende TDT4102-kommandoar for å opprette et fungerende kodeprosjekt:
     (a) `Ctrl+Shift+P` → `TDT4102: Force refresh of the course content`
     (b) `Ctrl+Shift+P` → `TDT4102: Create project from TDT4102 template` → `Configuration only`
4. Sørg for at du er inne i riktig fil i VS Code.
5. Kjør følgende TDT4102-kommandoer:
   (a) `Ctrl+Shift+P` → `TDT4102: Force refresh of the course content`
   (b) `Ctrl+Shift+P` → `TDT4102: Create project from TDT4102 template` → `Configuration only`
6. Prøv å kjøre koden igjen (`Ctrl+F5` eller `Fn+F5`).
7. Hvis det fortsatt ikke fungerer, lukk VS Code vinduet og åpne det igjen. Gjenta deretter steg 5-6.

Introduksjon

Nivåredigereren (`Level Editor`) er en enkel redigerer der brukeren kan lage en labyrint for et labyrintspill. Det utdelte kodeskjelettet mangler en del funksjonalitet. Den interne logikken foregår i et regulært to-dimensjonalt gitter (`grid`)). I zip-filen finner du kodeskjelettet med klart markerte oppgaver.
Før du starter må du sjekke at den umodifiserte utdelte koden kjører uten problemer. Du skal se det samme vinduet som i Figur 1.

Slik fungerer applikasjonen



Figur 1: Skjermbilde av kjøring av den utdelte koden.

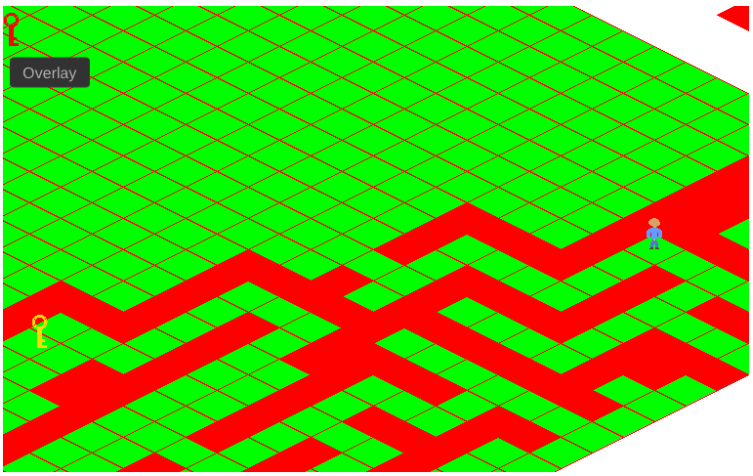
Figur 2: Skjermbilde av ferdig applikasjon.

Fra start ser verdenen i redigereren (figur 1) ganske tom ut. Lite grafikk og funksjonalitet er implementert. Når applikasjonen derimot er ferdig implementert (figur 2) er det fungerende med grafikk og logikk.

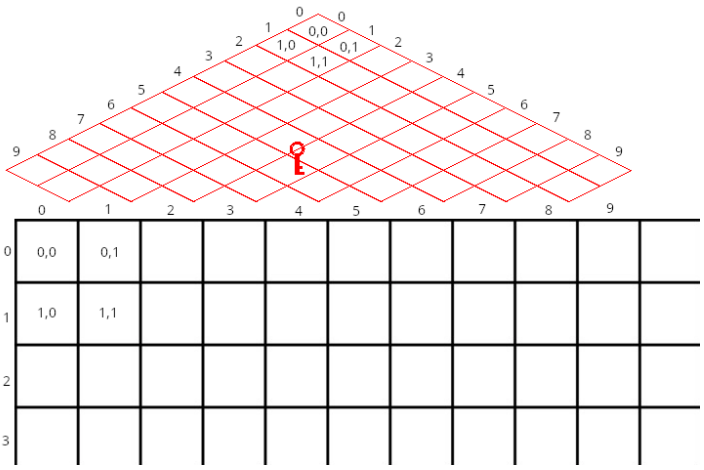
Det er når som helst mulig å anvende `Overlay`-knappen for å få en visuell representasjon av hvilke celler spilleren kan gå på. Cellene får en rød eller grønn farge avhengig av om cellen henholdsvis er blokkerte eller åpne for å gå på (figur 3).

Hvordan applikasjonen er satt opp

Tilstanden i verdenen håndteres gjennom `main.cpp`-filen. Denne filen skal **IKKE** røres. Det interne koordinatsystemet er et regulært todimensjonalt gitter. Gitt en verden med dimensjoner $w \times h$ vil cellen i posisjon (i, j) finnes på indeks $idx = j \times w + i$ (se figur 4).



Figur 3: Overlegget (`overlay`) når logikken for blokkerte og åpne celler er implementert.



Figur 4: Måten det visuelle gitteret samsvarer med det interne gitteret.

Hvordan besvare del 3?

Hver oppgave i del 3 har en tilhørende unik kode for å gjøre det lettere å finne frem til hvor du skal skrive svaret. Koden er på formatet `<T><siffer>` (*TS*), der sifrane er mellom 1 og 13 (*T1 - T13*). I `Tasks.cpp` vil du for hver oppgave finne to kommentarer som definerer henholdsvis begynnelsen og slutten av koden du skal føre inn. Kommentarene er på formatet: `// BEGIN: TS` og `// END: TS`.

Det er veldig viktig at alle svarene dine er skrevet mellom slike kommentarpar, for å støtte sensurmekanikken vår. Hvis det allerede er skrevet noen kode *mellom* `BEGIN`- og `END`-kommentarene i filene du har fått utdelt, så kan, og ofte bør, du erstatte den koden med din egen implementasjon med mindre annet er spesifisert i oppgavebeskrivelsen. All kode som står *utenfor* `BEGIN`- og `END`-kommentarene **SKAL** dere la stå. For eksempel, for oppgave *T1* ser du følgende kode i utdelte `Tasks.cpp`:

```
bool Level::is_walkable(const TDT4102::Point coordinate) const
{
    // BEGIN: T1
    // Write your answer to assignment T1 here, between the // BEGIN: T1
    // and // END: T1 comments. You should remove any code that is
    // already there and replace it with your own.

    // END: T1
}
```

Etter at du har implementert din løsning, bør du ende opp med følgende i stedet:

```
bool Level::is_walkable(const TDT4102::Point coordinate) const
{
    // BEGIN: T1
    // Write your answer to assignment T1 here, between the // BEGIN: T1
    // and // END: T1 comments. You should remove any code that is
    // already there and replace it with your own.

    /* Din kode her / Your code here */

    // END: T1
}
```

Merk at `BEGIN`- og `END`-kommentarene **IKKE** skal fjernes.

Oppgavene

Representere verdenen (30 poeng)

En verden representeres ved `Level`-klassen. Deklarasjonsfilen til denne finnes i `Level.hpp`. Klassen har en rekke funksjoner for å hente og endre tilstanden til verdenen. I denne delen skal du bare implementere funksjoner som lar utenforstående klasser hente tilstandsinformasjon fra verdenen.

```
class Level {
public:
    /* ... Constructors ... */
    bool is_walkable(const TDT4102::Point coordinate) const;
    int tile_at(const TDT4102::Point coordinate) const;
    void set_tile_at(const TDT4102::Point coordinate, const int tile);
    void set_walkable_at(const TDT4102::Point coordinate, const bool walkable);

    unsigned int get_width() const noexcept;
    unsigned int get_height() const noexcept;

private:
    unsigned int width = 1;
    unsigned int height = 1;

    std::vector<int> tiles = {0};
    std::vector<bool> walkable = {false};

    // ...
};
```

Legg merke til instansene av `std::vector` for `tiles` og `walkable`. Disse er fylt med informasjon om henholdsvis hvilke fliser som ligger i hver celle og hvorvidt cellen er mulig å gå på.

1. (5 points) **T1: Hente bredden av verdenen**

   Implementer koden for å hente ut det private feltet `width` i `Level`-klassen.

2. (5 points) **T2: Hente høyden av verdenen**

   Implementer koden for å hente ut det private feltet `height` i `Level`-klassen.

3. (10 points) **T3: Sjekk om ei flis har et tilhørende bilde**

   `Tile` klassen har et felt med navn `image`. Det er en delt peker (`shared pointer`) til en instans av `TDT4102::Image`.

   Skriv en funksjon som gir tilbake en boolsk verdi hvis pekeren er gyldig.

4. (10 points) **T4: Konstruktør for `Region`**

   Formålet med Region-`struct`en er å representere et område bestående av flere celler. Den har to felt: `begin` (start) og `end` (slutt), begge er instanser av `TDT4102::Point`. Konstruktøren skal tildele de laveste x- og y-verdiene blant de to punktene til `begin`-feltet, og de høyeste x- og y-verdiene til `end`-feltet.

   Skriv en implementasjon av denne konstruktøren.

Fikse kameraet (20 poeng)

`Context`-klassen

`Context`-klassen er en hjelperklasse som gir adgang til `AnimationWindow` and `Camera`-instansene som brukes i blant annet tegnekontekst. Metodene i klassen er `getWindow()` and `getCamera()` som henholdsvis returnerer referanser til `AnimationWindow` og `Camera`.

5. (20 points) **T5: Flytte på kameraet**

   Hittil har du stirret på den samme delen av nivået. Vi ønsker å gjøre kameraet funksjonelt slik at vi kan rotere kameraet horisontalt rundt. Hent referanser til instansene av `AnimationWindow` og `Camera` ved hjelp av `Context`-objektet, og bruk dem til **a)** å hente tastaturinndata og **b)** flytte kameraet basert på denne inndataen. En beskrivelse av hvordan funksjonen skal fungere følger nedenfor:

   Tabell 1: Tastene som brukes til å rotere kameraet horisontalt og hvordan de skal påvirke kameraet.

Tast	Effekt
W	Flytt kamera (`translate`) `-speed` i Y-retning
S	Flytt kamera (`translate`) `+speed` i Y-retning
A	Flytt kamera (`translate`) `-speed` i X-retning
D	Flytt kamera (`translate`) `+speed` i X-retning

Plassere fliser (60 poeng)

`Tile`-klassen

```
struct Tile
{
    Tile(int id, bool walkable, const std::filesystem::path tile_image_path);

    Tile(const Tile &other);
    Tile &operator=(const Tile &other);

    Tile(Tile &&rhs);
    Tile &operator=(Tile &&rhs);

    bool has_image() const noexcept;

    int id;
    bool walkable;
    std::shared_ptr<TDT4102::Image> image;
};
```

I tillegg til deklarasjonen av `Tile`-structen er det lurt å gjøre seg kjent med følgede to hjelpemetoder for tegning:

- `TileRenderer::render(Context &ctx, const Tile &tile, TDT4102::Point anchor)`
  - Tegner en flis på skjermkoordinatene oppgitt av `anchor`
- `QuadRenderer::render(Context &ctx, TDT4102::Point anchor, TDT4102::Color color)`
  - Tegner en quad på skjermkoordinatene oppgitt av `anchor` med oppgitt farge

Merk: For å unngå at programmet krasjer når man klikker utenfor nivået, sørg for at funksjonen ikke forsøker å skrive utenfor vektoren.

6. (10 points) **T6: Sett en flis**

   Koordinatene samsvarer med den interne koordinatrepresentasjonen som kan sees bakerst i dette dokumentet (s. 8). Et koordinat (x, y) i en verden som er $W \times H$ samsvarer med indeks $y \times W + x$.

   Endre på funksjonen som tar inn et koordinat og en flis-ID for at en flisen som er valgt skal okkupere den cellen som er klikket på. Skriv funksjonen slik at `tiles`-vektoren i `Level`-instansen reflekterer endringen.

7. (15 points) **T7: Sette en celle til å være åpen eller lukket**

   Skriv funksjonen slik at `walkable`-vektoren i `Level`-instansen setter et flagg for hvorvidt en celle er åpen eller lukket.

8. (15 points) **T8: Plasseringsoverlegg**

   Overlegget (`overlay`) for plassering (`placement`) av fliser burde allerede være synlig. Flisen som følger etter musepekeren er plasseringsoverlegget. Når du holder nede venstre musepeker vil du merke at flisen blir igjen der du først klikket fordi overlegget skal vise deg regionen du har valgt.

   Utvid funksjonen slik at den tegner samme flis over hele regionen som feltene `begin` og `end` spenner over.

9. (20 points) **T9: Oppdatere en region**

   Gitt de to punktne, `begin` and `end`, anvend den tidligere implenterte `set_tile_at` på hele regionen de to punktene spenner over.

Overlasting av operatorer og inn/ut datahåndteering (70 poeng)

`TileDescripto`-klassen

```
struct TileDescriptor {
    int id;
    std::string filename;
    bool walkable;
};
```

10. (15 points) **T10: Tilordningsoperator**

    Skriv operatoren for tilordning ved kopi slik at alle felter i `Tile` kopieres fra en eksisterende instans.

11. (15 points) **T11: Lese en enkelt beskrivelse av en flis**

    Skriv funksjonen `process_line` som skal ta imot en streng og konstruere en `TileDescriptor` fra den. Strukturen til en linje fra flisbeskriving-filen kan finnes bakerst i dette dokumentet (s. 8).

12. (20 points) **T12: Dynamisck innlasting av fliser**

    I `tiles/`-mappen finnes det en fil kalt `tile_info`. Denne beskriver en mengde fliser etter ID, filnavn på bildene til flisene og flagg for hvorvidt de er åpne eller lukkede (`walkable`).

    Skriv den overlastede operatoren slik at `TileDescriptor`-instansen fylles med data som sett i beskrivelsen av filtypen. Hvis funksjonen ikke kan åpne filen, kast en `runtime error` for å signalisere dette.

13. (20 points) **T13: Lagre nivået**

    Før du begynner på denne oppgaven, gjør deg kjent med filstrukturen for nivåer bakerst i dette dokumentet (s. 8).

    Implementer en lagrefunksjon for en `Level`-instans ved å åpne en filstrøm, kaste en feilmelding hvis filen ikke lar seg åpne, skrive overskriften bestående av bredden og høyden, skrive blokken med flis-IDer og skrive flaggblokken for åpne/lukkede celler.

    MERK: Hver linje begynner med en alfanumerisk karakter, dvs., det skal ikke være en tabulator på begynnelsen av en linje.

Nivå filstruktur

Et nivå lagres med følgende filformat:

- Den første linjen inneholder bredden og høyden til levelen, separert av en tabulator (`\t`).
- Etter dimensjonene følger en blokk med WI × HE heltall, som hver representerer en flis-ID.
- Blokken avsluttes med en enkelt linje som sier "END".
- Samme blokkoppsett brukes for å indikere hvorvidt en celle er åpen eller lukket. En celle er åpen hvis filen sier 1 og lukket hvis den er 0.

```
WI  HE
ID  ID  ID  ID  ID  ID  ID
ID  ID  ID  ID  ID  ID  ID
ID  ID  ID  ID  ID  ID  ID
ID  ID  ID  ID  ID  ID  ID
ID  ID  ID  ID  ID  ID  ID
END
WA  WA  WA  WA  WA  WA  WA
WA  WA  WA  WA  WA  WA  WA
WA  WA  WA  WA  WA  WA  WA
WA  WA  WA  WA  WA  WA  WA
WA  WA  WA  WA  WA  WA  WA
END
```

Flisbeskrivelse filstruktur

Hver linje av filen som beskriver fliser følger denne strukturen.

```
ID    IMAGE_PATH    WALKABLE

 0    house_00.png      0
 1    house_01.png      0
 2    house_02.png      1
 3    house_03.png      1
 4    house_04.png      1
10    house_10.png      0
20    house_20.png      0
28    house_28.png      0
...
```