Del III: Level Editor

Del 3 av eksamen er programmeringsoppgåver. Denne delen inneheld **13 oppgåver** som til saman gir ein maksimal poengsum på ca. 180 poeng og tel ca. **60 %** av eksamen. Kvar oppgåve gir maksimal poengsum på **5 - 20 poeng** avhengig av vanskegrad og arbeidsmengd.

Den utdelte koden inneheld kompilerbare (og kjørbare) *.cpp*- og *.h*-filar med forhåndskoda delar og ei full beskrivelse av oppgåvene i del 3 som ei PDF. Etter å ha lasta ned koden står du fritt til å bruke eit utviklingsmiljø (VS Code) for å jobbe med oppgåvene.

På neste side kan du laste ned *.zip*-fila om det **IKKJE** fungerer å henta ut utdelt kode via utvidinga til faget (extension), og seinere lasta opp den nye *.zip*-fila med din eigen kode inkludert.

Sjekkliste ved tekniske problem

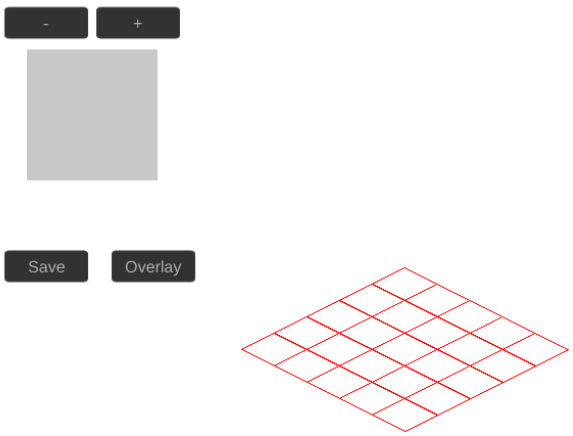
Viss prosjektet du opprettar med den utdelte koden ikkje kompilerer (før du har lagt til eigne endringer) bør du rekkje opp hånda og be om hjelp. Medan du venter kan du prøva følgjande:

1. Sjekk at prosjektmappa er lagret i mappa `C:\temp`.
2. Sjekk at namnet på mappa **IKKJE** inneheld norske bokstavar (*Æ, Ø, Å*) eller mellomrom. Dette kan skapa problem når du prøver å køyre koden i VS Code.
3. Om det **IKKJE** fungerer å henta ut utdelt kode via utvidinga til faget kan du prøve metoden som går ut på å laste ned zip-filen med utdelt kode frå Inspera:
   - Last ned .zip-filen frå Inspera og pakk den ut (unzip). Lagre fila i `C:\temp` som **eksamenTDT4102_kandidatnummer**. Du skal altså skriva inn *ditt eige kandidatnummer* bak understreken.
   - Opne mappa i VS Code. Bruk deretter følgjande TDT4102-kommandoar for å oppretta eit fungerande 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 rett fil i VS Code.
5. Køyr følgjande TDT4102-kommandoar:
   (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 køyra koden igjen (`Ctrl+F5` eller `Fn+F5`).
7. Viss det framleis ikkje fungerer, lukk VS Code vindauget og opne det igjen. Gjenta deretter steg 5-6.

Introduksjon

Nivåredigeraren (`Level Editor`) er ein enkel redigerar der brukaren kan laga ein labyrint for eit labyringspill. Den interne logikken går føre seg i eit regulært to-dimensjonalt gitter (`grid`). Det utdelte kodeskjelettet manglar ein del funksjonalitet. I zip-fila finn du kodeskjelettet med klart markerte oppgåver.
Før du startar må du sjekka at den (umodifiserte) utdelte koden køyrer utan problem. Du skal sjå det same vindauget som i Figur 1.

Slik fungerer applikasjonen



Figur 1: Skjermbilde av køyring av den utdelte koden.

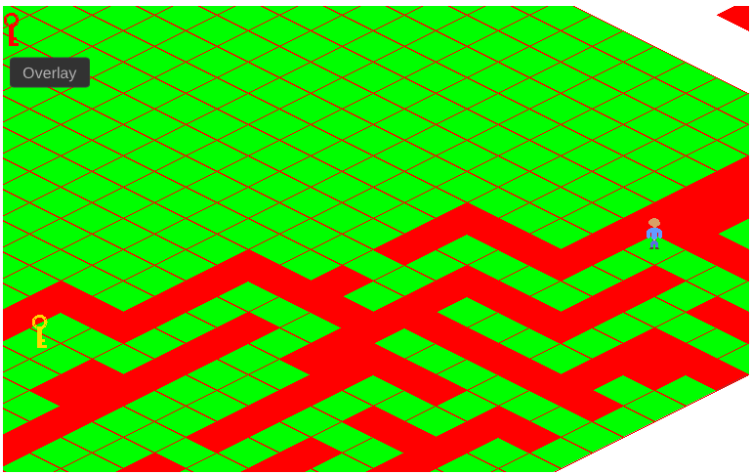
Figur 2: Skjermbilde av ferdig applikasjon.

Frå start ser redigeraren (figur 1) ganske tom ut. Lite grafikk og funksjonalitet er implementert. Når applikasjonen derimot er ferdig implementert (figur 2) er det fungerand med grafikk og logikk.

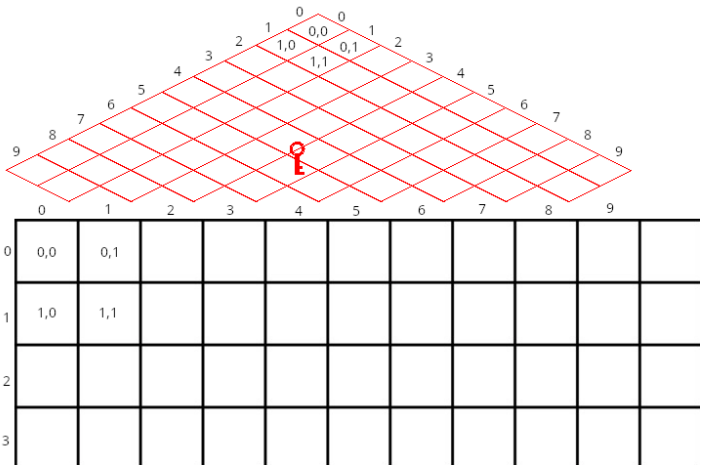
Det er når som helst mogleg å nytta `Overlay`-knappen for å få ein visuell representasjon av kva celler spelaren kan gå på.Cellene får ein raud eller grøn farge avhengig av om cella høvesvis er blokkerte eller opne for å gå på (figur 3).

Korleis applikasjonen er sett opp

Tilstanden i verda blir handtert gjennom `main.cpp`-fila. Denne fila skal **IKKJE** rørast. Det interne koordinatsystemet er eit regulært todimensjonalt gitter. Gitt ei verd med dimensjonar $w \times h$ vil cella i posisjon (i, j) finnast på indeks $idx = j \times w + i$ (Sjå figur 4).



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



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

Korleis svara på del 3?

Kvar oppgåve i del 3 har ein tilhøyrande unik kode for å gjera det lettare å finna fram til kor du skal skriva svaret. Koden er på formata `<T><siffer>` (*TS*), der sifrane er mellom 1 og 13 (*T1* - *T13*). I `Tasks.cpp` vil du for kvar oppgåve finna to kommentarar som definerer høvesvis byrjinga og slutten av koden du skal føra inn. Kommentarane er på formatet: `// BEGIN: TS` og `// END: TS`.

Det er veldig viktig at alle svara dine er skrivne mellom slike kommentarpar, for å støtta sensurmekanikken vår. Viss det allereie er skrive noko kode *mellom* `BEGIN`- og `END`-kommentarane i filene du har fått utdelt, så kan, og ofte bør, du erstatta den koden med din eigen implementasjon med mindre anna er spesifisert i oppgåveskildringa. All kode som står *utanfor* `BEGIN`- og `END`-kommentarane **SKAL** de la stå.
Til dømes, for oppgåve *T1* ser du følgjande 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 løysinga di, bør du enda opp med følgjande i staden:

```
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`-kommentarane **IKKJE** skal fjernast.

Oppgåvene

Representere verda (30 poeng)

Ei verd blir representert ved `Level`-klassen. Deklarasjonsfila til denne finst i `Level.hpp`. Klassen har ei rekkje funksjonar for å henta og endra tilstanden til verda. I denne delen skal du berre implementera funksjonar som lèt utanforståande klassar henta tilstandsinformasjon frå verda.

```
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 instansane av `std::vector` for `tiles` og `walkable`. Desse er fylte med informasjon om høvesvis kva fliser som ligg i kvar celle og om cella er mogleg å gå på.

1. (5 points) **T1: Hente bredda av verda**

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

2. (5 points) **T2: Hente høgda av verda**

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

3. (10 points) **T3: Sjekk om ei flis har eit tilhørande bilete**

   `Tile` klassa har eit felt namngitt `image`. Det er ein delt peikar (`shared pointer`) til ein instans av `TDT4102::Image`.

   Skriv ein funksjon som gjev tilbake ein boolsk verdi hvis peikaren er gyldig.

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

   Formålet med Region-`structen` er å representere eit område bestående av fleire cellar. Han har to felt: `begin` (start) og `end` (slutt), begge er instansar av `TDT4102::Point`. Konstruktøren skal tildele dei lågaste x- og y-verdiane blant dei to punkta til `begin`-feltet, og dei høgaste x- og y-verdiane til `end`-feltet.

   Skriv ei implementasjon av denne konstruktøren.

Fikse kameraet (20 poeng)

`Context`-klassen

`Context`-klassen er ein hjelparklasse som gir tilgang til `AnimationWindow` and `Camera`-instansane som blir brukte i mellom anna teiknekontekst. Metodane i klassen er `getWindow()` and `getCamera()` som høvesvis returnerer referansar til `AnimationWindow` og `Camera`.

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

   Så langt har du stirt på den same delen av nivået. Me ønskjer å gjera kameraet funksjonelt slik at me kan rotera kameraet horisontalt rundt. Hent referansar til instansane av `AnimationWindow` og `Camera` ved hjelp av `Context`-objektet, og bruk dei til **a)** å henta tastaturinndata og **b)** flotne kameraet basert på dette inndataet. Ei skildring av korleis funksjonen skal fungera følgjar nedanfor:

   Tabell 1: Tastane som blir brukte til å rotera kameraet horisontalt og korleis dei skal påverka 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 gjera seg kjend med følgjande to hjelpmetodane for å teikne:

- `TileRenderer::render(Context &ctx, const Tile &tile, TDT4102::Point anchor)`
  - Teiknar ei flis på skjermkoordinatana oppgitt av `anchor`
- `QuadRenderer::render(Context &ctx, TDT4102::Point anchor, TDT4102::Color color)`
  - Teiknar ein quad på skjermkoordinatana oppgitt av `anchor` med oppgitt farge

Merk: For å unngå at programmet krasjar når ein klikkar utanfor nivået, sørg for at funksjonen ikkje prøver å skriva utanfor vektoren.

6. (10 points) **T6: Sette ein flis**

   Koordinatane samsvarer med den interne koordinatrepresentasjonen som kan sjåast bakarst i dette dokumentet (s. 8). Eit koordinat (x, y) i ein verd som er $W \times H$ samsvarer med indeks $y \times W + x$.

   Endre på funksjonen som tek inn eit koordinat og ein flis-ID for at ein flis som er vald skal okkupera den cella som er klikka på. Skriv funksjonen slik at `tiles`-vektoren i `Level`-instansen reflekterer endringa.

7. (15 points) **T7: Sette ei celle til å vere open eller lukka**

   Skriv funksjonen slik at `walkable`-vektoren i `Level`-instansen set eit flagg for om ei celle er open eller lukka.

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

   Overlegget (`overlay`) for plassering (`placement`) av fliser burde allereie vera synleg. Flisa som følgjer etter musepeikaren er plasseringsoverlegget. Når du held nede venstre musepeikar vil du merka at flisa blir igjen der du først klikka fordi overlegget skal visa deg regionen du har valt.

   Utvid funksjonen slik at den teiknaren same flis over heile regionen som felta `begin` og `end` spenner over.

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

   Gitt de to punktene, `begin` and `end`, nytt den tidligare implenterte `set_tile_at` på heile regionen dei to punkta spenner over.

Overlasting av operatorer og inn/ut datahandtering (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 felt i `Tile` blir kopierte frå ein eksisterande instans.

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

    FSkriv fnksjonen `process_line` som skal ta imot ein streng og konstruera ein `TileDescriptor` frå han. Strukturen til ei linje frå flisbeskrivelse-fila kan finnast bakarst i dette dokumentet (s. 8).

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

    I `tiles/`-mappa finst det ei fil kalla `tile_info`. Denne beskriv ei mengde fliser etter ID, filnamn på bileta til flisene og flagg for om dei er opne eller lukka (`walkable`).

    Skriv den overlasta operatoren slik at `TileDescriptor`-instansen blir fylt med data som sett i skildringa av filtypen. Viss funksjonen ikkje kan opna fila, kast ein `runtime error` for å signalisera dette.

13. (20 points) **T13: Lagre nivået**

    Før du byrjar på denne oppgåva, gjer deg kjent med filstrukturen for nivå bakarst i dette dokumentet (s. 8).

    Implementer ein lagrefunksjon for ein `Level`-instans ved å opne ein filstraum, kaste ein error viss fila ikkje lèt seg opna, skriva overskrifta beståande av breidda og høgda, skriva blokka med flis-IDar og skriva flaggblokka for opne/lukka celler.

    MERK: Kvar linje byrjar med ein alfanumerisk karakter, dvs., det skal ikkje vera ein tabulator på byrjinga av ei 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
```

Flisbeskriving 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
...
```