Del III: Maze Game (180p)

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

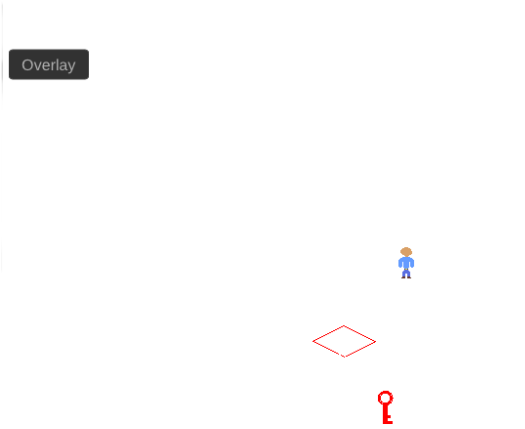
Viss prosjektet du opprettar med den utdelte koden ikkje kompilerer (før du har lagt til eigne endringer) bør du rekkje opp hånda og be om hjelp. Medan du venter kan du prøva følgjande:

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.

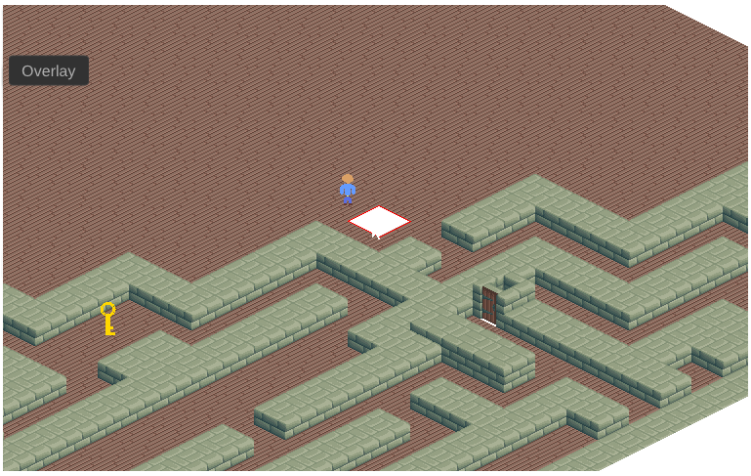
Introduksjon

Labyrintspelet (*Maze Game*) er eit enkelt spel der spelaren får oppgåve i å finne nøklar, låse opp dørar og finne vegen ut. Den overrekte koden mangler ein del funksjonar. Den interne logikken i spelet føregår i eit regulært todimensjonalt gitter (`grid`). 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.



Figur 1: Skjermbilde av køyring av den utdelte koden.



Figur 2: Skjermbilde av det ferdige spelet.

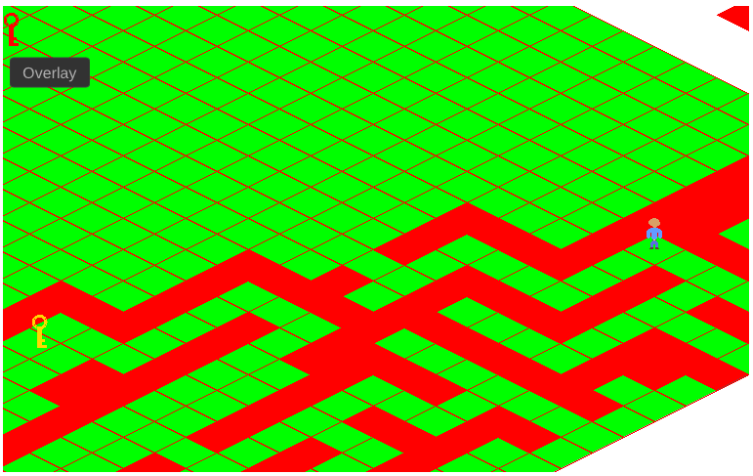
Slik fungerer Labyrintspelet

Frå start ser verda i Labyrintspillet (figur 1) ganske tom ut. Lite grafikk og funksjonalitet er implementert. Når spelet derimot er ferdig implementert (figur 2) er det spelebart med grafikk og logikk.

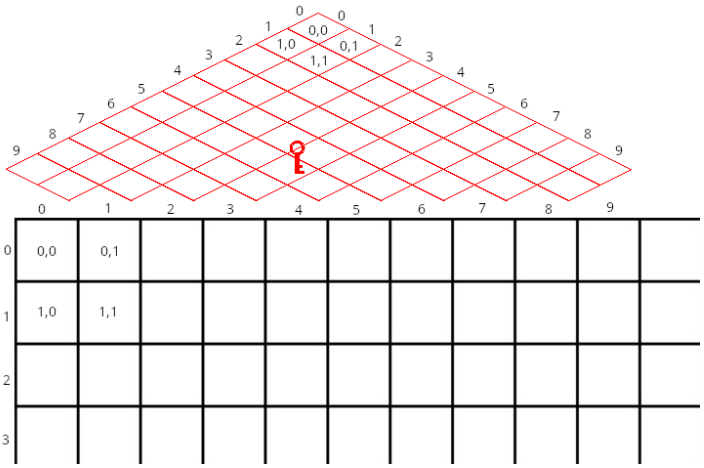
Det er når som helst mogleg å nytta `Overlay`-knappen for å få ein visuell representasjon av kva celler spelaren kan gå på. Etter logikken er implementert vil cellene få ein raud eller grøn farge avhengig av om cella høvesvis er blokkerte eller opne for å gå på (figur 3).

Korleis Labyrintspelet 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 wh vil cella i posisjon (i, j) finnast på indeks $idx = jw + i$ (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 `Tasks.h`. 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 celle er open for å gå på.**

   Finn ut av om det er mogleg å gå på ei gitt celle gitt informasjonen i `walkable`-feltet og ein `Point`-parameter namngitt `coordinate`. `coordinate`-parametret følgjer alltid det interne koordinatsystemet gitt i figur 4. Returverdien av funksjonen skal vera ein boolsk verdi som oppgir om der er mogleg å gå på cella.

4. (10 points) **T4: Finn ut av kva flis som ligg i ei gitt celle**

   Finn ut av kva type flis som opptek cella, gitt informasjonen i `tiles`-feltet og `Point`-parameteren `coordinate`. Returverdien av funksjonen skal vera ein heiltalsverdi som oppgir ID-en til flisa.

   Obs! Koden du skriv her vil ikkje vera mogleg å testa visuelt før oppgåve *T6* er gjord.

Teikning av flisane (25 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;
};
```

`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. (10 points) **T5: Kopikonstruktør for** `Tile`

   For å kunna teikna fliser, må me kunna senda gyldige instansar av `Tile` til teiknelogikken. Akkurat no gjer ikkje kopikonstruktøren det han skal gjera. Alle kopikonstruerte objekt endar opp i `default`-initialisert tilstand. Fyll ut kopikonstruktøren slik at felta blir kopierte på rett måte.

   Obs! Koden du skriv her vil ikkje vera mogleg å testa visuelt før oppgåve *T6* er gjord.

6. (15 points) **T6: Teikne flisar**

   Skriv koden for å teikna ei flis på staden gitt av `anchor`-parameteren. Dette angir eit koordinat på **skjermen** og ikkje i det interne gitteret. `AnimationWindow` instansen må hentast via `Context` klassen. Unngå å teikna viss flisa (`tile`) ikkje inneheld eit bilete.

   Merk: Berre ein type flis er tilgjengeleg fram til oppgåve *T11* er gjord. Bruk `Overlay`-knappen for å visualisera spellogikken.

Flytte på ting (55 poeng)

Eit spel utan rørsle eller motiverande faktorar er eit kjedeleg spel. På dette punktet skal Labyrintspillet ha grafikk, men det er veldig lite som skjer. Her skal du sørgja for at spelaren berre kan bevega seg til opne celler og for at spelaren kan flytta på kameraet.

7. (15 points) **T7: Avgrensa rørsla til spelaren**

   Implementer koden for å verifisera ei foreslått spelarrørsle. Den noverande spelarposisjonen kan lesast frå `position`-feltet i klassen `PlayerControllable`. Her er ei liste over krava for at ei rørsle skal vera gyldig:

   - Den nye posisjonen er innanfor verdas rammar ($0 \leq x \leq w$ og $0 \leq y \leq h$).
   - Spelaren bevegar seg ikkje lengre enn 1 lengdeeining ($|x1 - x0| \leq 1$).
   - Målcella er mogleg å gå på.

   Returverdien skal vera ein boolsk verdi som seier om rørsla er gyldig.

8. (20 points) **T8: 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

9. (20 points) **T9: Flytta på spelaren**

   Spelaren kan allereie flytta på seg ved å klikka på celler. Bruk `AnimationWindow`-argumentet og bruk han til **a)** å henta tastaturinndata og **b)** flotne spelaren basert på dette inndataet. Ei skildring av korleis funksjonen skal fungera følgjar nedanfor:

   Sørg for at rørsla er gyldig før spelaren flyttar seg med `moveTo`-metoden. Både `canMoveTo` og `moveTo` tek inn **1)** eit `Point`-objekt som representerer eit koordinat og **2)** ein instans av `Level`.

Inndata / Utdata (70 poeng)

Så langt har du jobba med eit statisk nivå som berre har éin flistype. Ideelt sett ønskjer me variasjon for eit spel som dette. Her skal du implementera noko av funksjonaliteten for å lasta nivå og fliser.

Tabell 2: Tastane som blir brukte til å rotera kameraet horisontalt og korleis dei skal påverka kameraet.

Tast	Effekt
`ARROW_UP`	Flytt spilleren -1 i Y-retning
`ARROW_DOWN`	Flytt spilleren +1 i Y-retning
`ARROW_LEFT`	Flytt spilleren -1 i X-retning
`ARROW_RIGHT`	Flytt spilleren +1 i X-retning

10. (15 points) **T10: Overlasting av uthentingsoperatoren for fliser**

    Som del av innlasting av fliser bidreg uthentingsoperatoren til å retta ein input-straum på eit `TileDescriptor` objekt. `TileDescriptor` formålet sitt er å gi ei skildring av ei flis som kan lastast inn. Ei skildring av filtypen som listar opp fliser og skildringane deira finst bakarst i dette dokumentet (s. 8).

    Skriv den overlasta operatoren slik at `TileDescriptor` instansen blir fylt med data som sett i skildringa av filtypen.

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

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

    Som del av innlasting av fliser bidreg uthentingsoperatoren til å retta ein input-straum på eit `TileDescriptor` objekt.

    Skriv den overlasta operatoren slik at `TileDescriptor` instansen blir fylt med data som gitt i skildringa av filtypen.

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

12. (20 points) **T12: Innlasting av nivå**

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

    Funksjonen du skal jobba med i denne oppgåva tek inn ein sti til ei fil som beskriv eit nivå. Resultatet skal vera ein instans av `Level` som er konstruerte og fylte med data tilsvarande dei du finn i fila. Viss funksjonen feilar skal den kasta ein feilmelding (`runtime error`) slik som til dømes *"Could not load [filename]"*.

13. (20 points) **T13: Sammenligningsoperator for koordinater**

    Implementer samanlikningsoperatoren for `Point` klassen.

Nivå filstruktur

Eit nivå blir lagra med følgjande filformat:

- Den første linja inneheld breidda og høgda til nivået, separert av ein tabulator(\t).
- Etter dimensjonane følgjer ei blokk med WI × HE heiltal, som kvar representerer ein flis-ID.
- Blokka blir avslutta med ei enkelt linje som seier "END".
- Same blokkoppsett blir brukt for å indikera om ei celle er open eller lukka. Ei celle er open viss fila seier 1 og lukka viss ho 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

Kvar linje av fila som beskriv fliser følgjer 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
...
```