# Del III: Maze Game (180p)

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

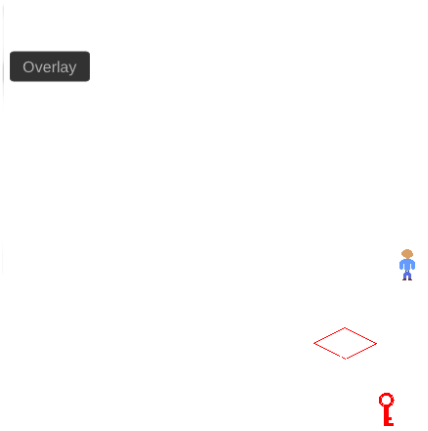
Hvis prosjektet du oppretter med den utdelte koden ikke kompilerer (før du har lagt til egne endringer) bør du rekke opp hånden og be om hjelp. Mens du venter kan du prøve følgende:

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.

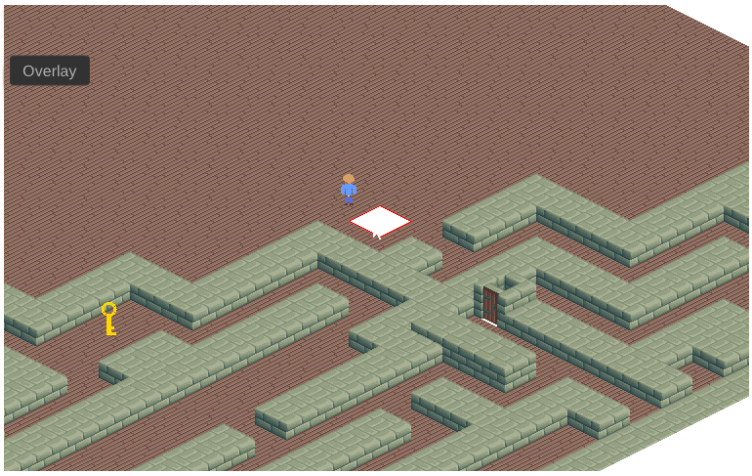
# Introduksjon

Labyrintspillet (*Maze Game*) er et enkelt spill der spilleren får i oppgave å finne nøkler, låse opp dører og finne veien ut. Den utdelte koden mangler en del funksjonalitet. Den interne logikken i spillet foregår i et regulært todimensjonalt 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.




**Figur 1:** Skjermbilde av kjøring av den utdelte koden.




**Figur 2:** Skjermbilde av ferdig spill.

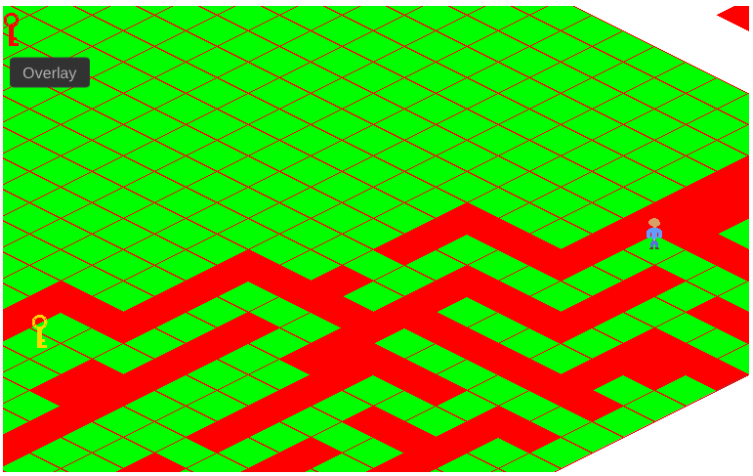
## Slik fungerer Labyrintspillet

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

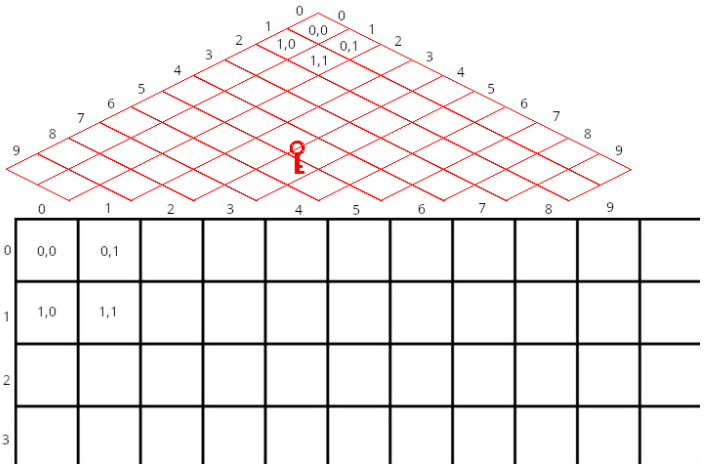
Det er når som helst mulig å anvende `Overlay`-knappen for å få en visuell representasjon av hvilke celler spilleren kan gå på. Før logikken er implementert vil alle celler fylles med en rød farge. Etter logikken er implementert vil cellene få en rød eller grønn farge avhengig av om cellen henholdsvis er blokkerte eller åpne for å gå på (figur 3).

## Hvordan Labyrintspillet 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 sifrene er mellom 1 og 13 (*T1* - *T13*). I `Tasks.cpp` vil man 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 `Tasks.h`. 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 celle er åpen for å gå på**

   Finn ut av hvorvidt det er mulig å gå på en gitt celle gitt informasjonen i `walkable`-feltet og en `Point`-parameter kalt `coordinate`. `coordinate`-parametret følger alltid det interne koordinatsystemet gitt i figur 4. Returverdien av funksjonen skal være en boolsk verdi som oppgir hvorvidt der er mulig å gå på cellen.

4. (10 points) **T4: Finn ut av hvilken flis som ligger i en gitt celle**

   Finn ut av hvilken type flis som opptar cellen, gitt informasjonen i `tiles`-feltet og `Point`-parameteren `coordinate`. Returverdien av funksjonen skal være en heltallsverdi som oppgir flisens ID.

   **NB!** Koden du skriver her vil ikke være mulig å teste visuelt før oppgave *T6* er gjort.

## Tegning av flisene (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 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. (10 points) **T5: Kopikonstruktør for** `Tile`

   For å kunne tegne fliser, må vi kunne sende gyldige instanser av `Tile` til tegnelogikken. For øyeblikket gjør ikke kopikonstruktøren det den skal gjøre. Alle kopikonstruerte objekter ender opp i `default`-initialisert tilstand. Fyll ut kopikonstruktøren slik at feltene blir kopiert på riktig måte.

   **NB!** Koden du skriver her vil ikke være mulig å teste visuelt før oppgave *T6* er gjort.

6. (15 points) **T6: Tegne fliser**

   Skriv koden for å tegne en flis på stedet gitt av `anchor`-parameteren. Dette angir et koordinat på **skjermen** og ikke i det interne gitteret. `AnimationWindow` instansen må hentes via `Context` klassen. Unngå å tegne hvis flisen (`tile`) ikke inneholder et bilde.

   **Merk:** Bare én type flis er tilgjengelig frem til oppgave *T11* er gjort. Bruk `Overlay`-knappen for å visualisere spillogikken.

## Flytte på ting (55 poeng)

Et spill uten bevegelse eller motiverende faktorer er et kjedelig spill. På dette punktet skal Labyrintspillet ha grafikk, men det er veldig lite som skjer. Her skal du sørge for at spilleren kun kan bevege seg til åpne celler og for at spilleren kan flytte på kameraet.

7. (15 points) **T7: Begrense bevegeligheten til spilleren**

   Implementer koden for å verifisere en foreslått spillerbevegelse. Den nåværende spillerposisjonen kan leses fra `position`-feltet i klassen `PlayerControllable`. Her er en liste over kravene for at en bevegelse skal være gyldig:

   - Den nye posisjonen er innenfor verdenens rammer ($0 \leq x \leq w$ og $0 \leq y \leq h$ ).
   - Spilleren beveger seg ikke lengre enn 1 lengdeenhet ($|x1 - x0| \leq 1$).
   - Målcellen er mulig å gå på.

   Returverdien skal være en boolsk verdi som sier hvorvidt bevegelsen er gyldig.

8. (20 points) **T8: 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 |

9. (20 points) **T9: Flytte på spilleren**

   Spilleren kan allerede flytte på seg ved å klikke på celler. Bruk `AnimationWindow`-argumentet og bruk den til **a)** å hente tastaturinndata og **b)** flytte spilleren basert på denne inndataen. En beskrivelse av hvordan funksjonen skal fungere følger nedenfor:

   Sørg for at bevegelsen er gyldig før spilleren flytter seg med `moveTo`-metoden. Både `canMoveTo` og `moveTo` tar inn **1)** et `Point`-objekt som representerer et koordinat og **2)** en instans av `Level`.

## Inndata / Utdata (70 poeng)

Så langt har du jobbet med et statisk nivå som bare har én flistype. Ideelt sett ønsker vi variasjon for et spill som dette. Her skal du implementere noe av funksjonaliteten for å laste nivåer og fliser.

**Tabell 2:** Tastene som brukes til å rotere kameraet horisontalt og hvordan de skal påvirke 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 over fliser**

    Som del av innlasting av fliser bidrar uthentingsoperatoren til å rette en input-strøm på et `TileDescriptor`-objekt. `TileDescriptor` sitt formål er å gi en beskrivelse av en flis som kan lastes inn. En beskrivelse av filtypen som lister opp fliser og deres beskrivelser finnes bakerst i dette dokumentet (s. 8).

    Skriv den overlastede operatoren slik at `TileDescriptor` instansen fylles med data som sett i beskrivelsen 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 bidrar uthentingsoperatoren til å rette en input-strøm på et `TileDescriptor` objekt.

    Skriv den overlastede operatoren slik at `TileDescriptor` instansen fylles med data som gitt i beskrivelsen av filtypen.

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

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

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

    Funksjonen du skal jobbe med i denne oppgaven tar inn en sti til en fil som beskriver et nivå. Resultatet skal være en instans av `Level` som er konstruert og fylt med data tilsvarende de du finner i filen. Hvis funksjonen feiler skal den kaste en feilmelding (`runtime error`) slik som for eksempel *"Could not load [filename]"*.

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

    Implementer sammenligningsoperatoren for `Point` klassen.

## Nivå filstruktur

Et nivå lagres med følgende filformat:

- Den første linjen inneholder bredden og høyden til nivået, 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
...
```