Lær å kode i Unity – del 2: Logiske uttrykk, relasjonsoperatører og if-setninger

Har du noen gang tenkt over koden som ligger bak valgene i et spill? Da tenker jeg ikke bare på story-relaterte spill hvor et valg du tar underveis gjør at du får en helt annen ending enn om du hadde tatt det andre valget de gav deg på det tidspunktet. Jeg tenker også på «hva skjer om jeg velger å trykke «play»-knappen på skjermen nå?», «hva skjer om jeg prøver å pukke opp en gjenstand dersom baggen min allerede er full?», «Hva skjer dersom jeg har 3 liv og mister 1 liv?». Vi som programmerer spillet velger selv hva som skjer i de ulike scenarioene. Vi kan for eksempel velge at hvis spilleren prøver å plukke opp en gjenstand dersom baggen allerede er full, så skal det gis en beskjed i form av tekst på skjermen til spilleren om at dette ikke er mulig. For å få programmet/spillet til å utføre disse instruksene må vi kunne teste om valget blir tatt. «Trykket spilleren på «play»-knappen? Ja: Start spillet. Nei: Fortsett å spille av bakgrunnsmusikk/animasjon i menyen.» For å gjøre dette trenger vi blant annet å kunne noe om logiske utrykk, relasjonsoperatører og if-setninger.

Logiske uttrykk og relasjonsoperatører

Et logisk uttrykk er en «setning» som har en verdi true (sann) eller false (usann). På vårt norske menneskespråk kan dette skrives f.eks. i form av et utsagn: «Katter lander alltid på beina!» (verdi sann eller usann) eller f.eks. et regnestykke «1 + 1 = 2» (verdi sann eller usann). Den enkleste formen for et logisk uttrykk innenfor programmering er en «logisk variabel». Dette er en variabel av datatypen «boolean» og kan skrives slik:

```
bool gameOver = false;
```

bool

Logiske uttrykk og logiske variabler brukes til å skille mellom forskjellige alternativer for valg inne i programmet dersom det finnes flere mulige valg til et scenario. Vi kan blant annet sammenlikne tallstørrelser, eller for eksempel sjekke om en tellevariabel er mindre enn en bestemt grenseverdi. Måten vi skriver disse eksemplene på er linkende slik vi gjør det i matematikken, men kodespråkene har sin egen lille vri på hvordan det skrives. Her er noen eksempler på det vi kaller «relasjonsoperatorer»:

x == y «er lik»
x != y «er ikke lik»
x < y «er mindre enn»
x > y «er større enn»
x <= y «er mindre enn eller lik»
x >= y «er større enn eller lik»

Felles for alle disse utrykkene er at de kan enten være sanne (true) eller usanne (false). Merk at de operatorene som består at to separate tegn må skrives uten mellomrom! Det er heller ikke tillatt å f.eks. skrive =! i stedet for != eller =< i stedet for <=. En huskeregel er at = skal alltid stå til slutt i disse sammensetningene.

Det kan også være viktig å merke seg at operatorene == og != har lavere prioritet enn de andre relasjonsoperatorene. Det betyr at dersom flere operatorer forekommer i ett og samme uttrykk, vil == og != bli utført først *etter* at de andre operatorene er utført.

Oppgave:

Tenk deg at vi har variablene m og n hvor verdiene er som følgende:

```
int m = 5;

int n = -7;
```

Avgjør om følgende uttrykk har verdien true eller false:

```
m < n

n >= m

n <= m

n > m

m != n

m == n
```

If-setninger

Når du lager et spill er det veldig viktig å kunne lage valgmuligheter inne i selve koden din. Vi bruker if-setninger for å uttrykke at en instruks bare skal utføres dersom en bestemt betingelse er oppfylt. Tenk deg f.eks. at du ønsker å skrive ut en tekst «Game Over!» dersom spillet ditt er over. For å vite om dette er sant må vi lage kode som tester dette.

If-setninger brukes for å teste om en betingelse er sann eller usann og skrives på følgende måte:

if (betingelse) {

```
    instuksjon
```

}

Betingelsen må være et logisk uttrykk (dvs. C# kode som har verdi true eller false).

Instruksjonen(e) er C# kode som vi ønsker utført dersom betingelsen har verdi sann/true. Dersom betingelsen har verdi usann/false vil instruksjonen(e) ikke bli utført.

Lag et nytt script og kall det «Conditionals».

(Husk å dra scriptet over til «Main Camera» i Unity. Pass på at navnet du gir scriptet i Unity stemmer overens med navnet i selve scriptet.)

NB! Når du skriver kode kan det viktig å kommentere hva du gjør underveis, spesielt hvis flere personer samarbeider med et prosjekt og prosjektet er stort (Spill som f.eks. world of warcraft har over 10 millioner linjer med kode!). Kanskje må person 2 bruke en bestemt del av koden til person 1 i sin del av koden og trenger å forstå hva person 2 mente da han lagde variablen «int abcdefg = 23;». Kommentarer skrives på følgende måte:

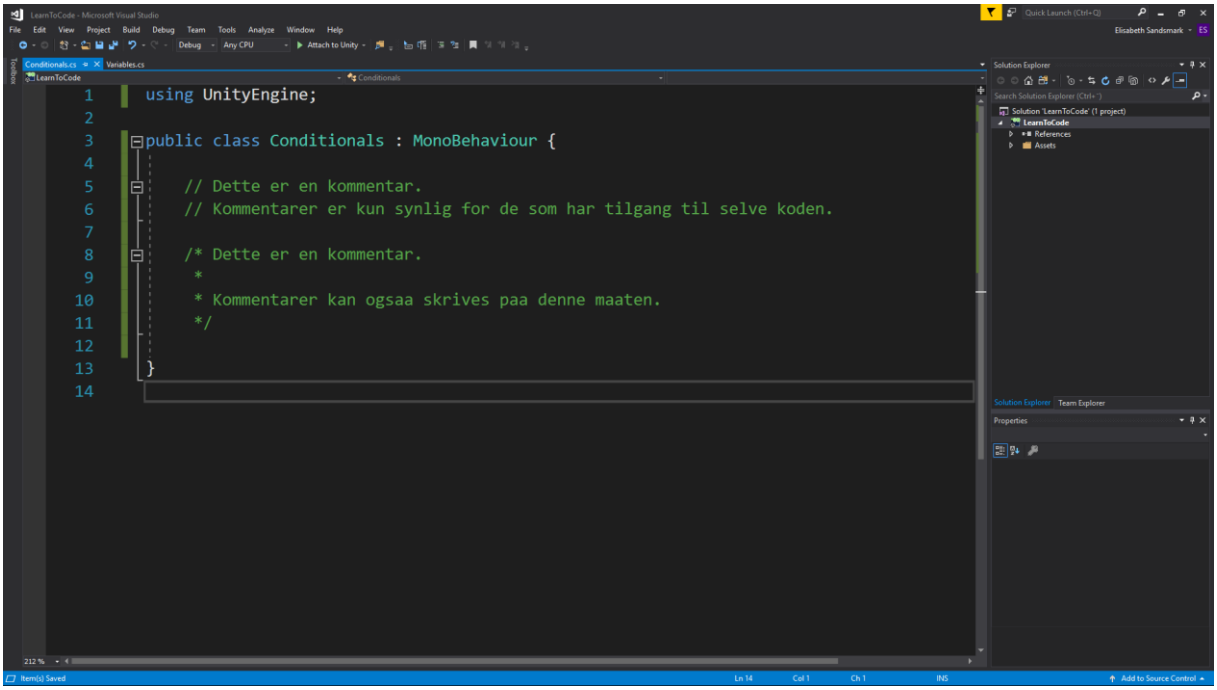
// skriv kommentar her.

Eller slik:

/* Skriv en kommentar her. */

Metode 1 brukes mest for korte kommentarer som skrives på en eller få linjer, mens metode 2 brukes for lengre kommentarer som går over mange linjer og startes med /* og avsluttes med */.

Kommentaren vil kun være synlig for de som programmerer. *(Der viktig å huske at det ikke er mulig å bruke norske bokstaver som æ, ø eller å hverken i kommentarer eller i andre deler av koden din. Det kan hende at du kan kjøre koden på din PC, men det betyr ikke at det fungerer for andre!)*



Eksempel 1

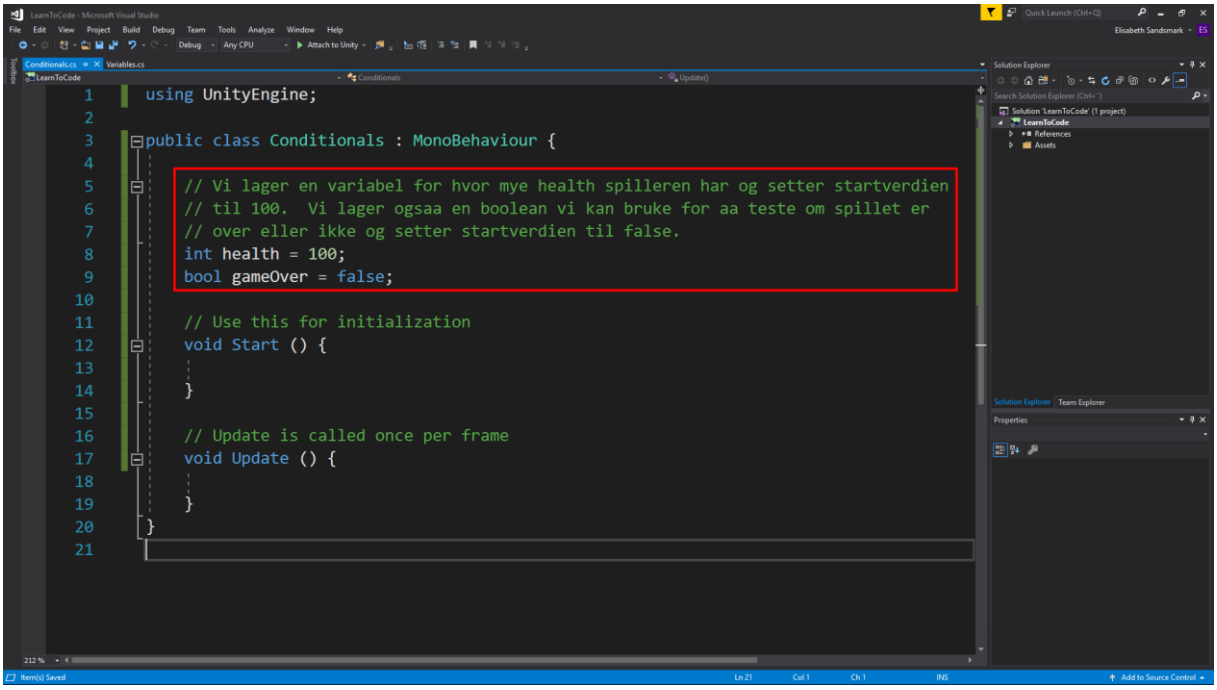
I dette eksemplet bruker vi booleans og if-setninger til å teste om spillerens «health» er mindre eller lik 0 og gir instruksjoner til hva som skal skje ut ifra om det som returneres er true eller false.

(Du trenger ikke å skrive inn kommentarene dersom du ikke vil. De er der bare for å beskrive hva som skjer i koden og er for så vidt forholdsvis unødvendige i små programmer etter at du har fått enn grunnleggende forståelse for hvordan koding fungerer. For mange kommentarer gjør at det fort kan se rotete ut.)

Vi begynner med å lage noen variabler som vi kan bruke for å teste if-setningene våre:

// Vi lager en variabel for hvor mye health spilleren har og setter startverdien til 100. Vi lager ogsaa en boolean vi kan bruke for aa teste om spillet er over eller ikke og setter startverdien til false.

```
int health = 100;

bool gameOver = false;
```



Deretter kan vi begynne på if-setningene våre:

// Tester om spillerens health er mindre eller lik 0. Dersom dette stemmer blir variablen «gameOver» sann.

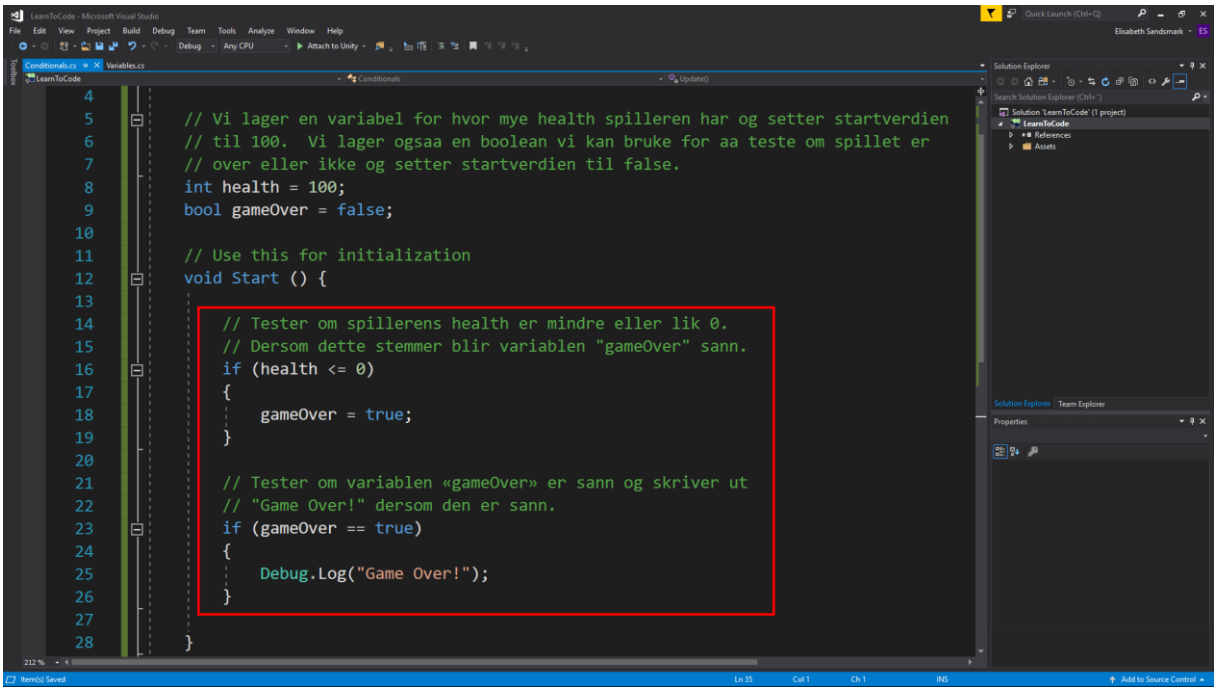
```
if (health <= 0) {

        gameOver = true;

}
```

// Tester om variablen «gameOver» er sann og skriver ut «Game Over!» dersom den er sann.

if (gameOver == true) {

Debug.Log («Game Over!»);

}



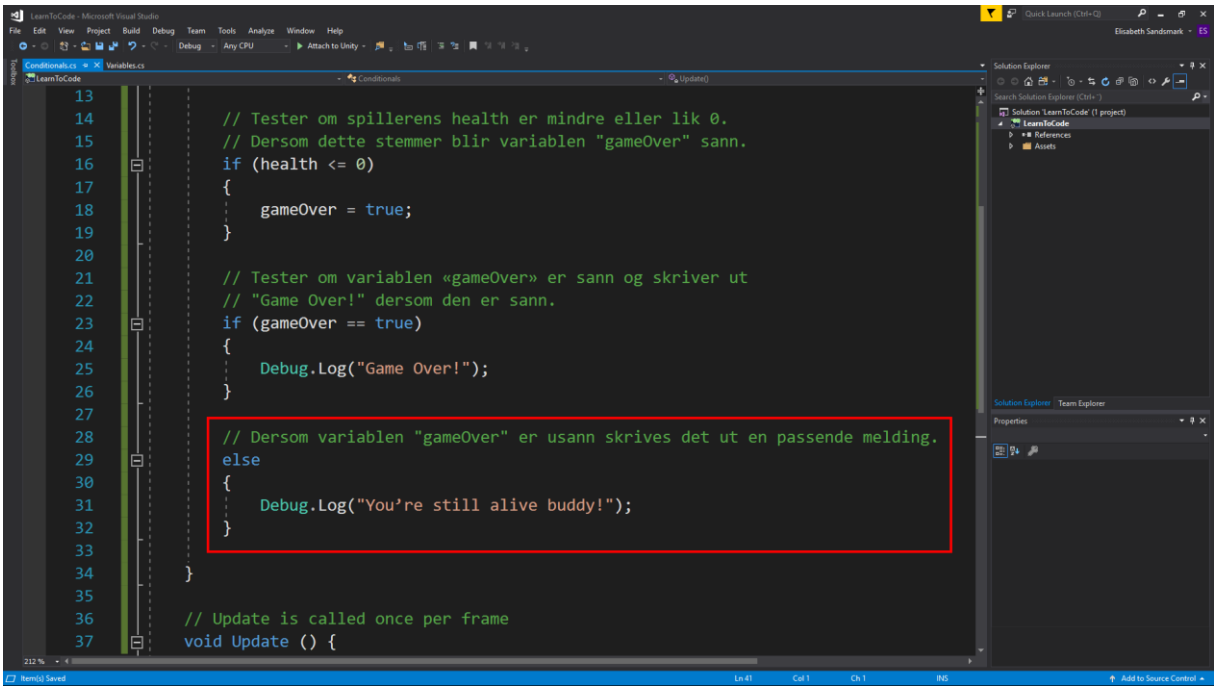
Det er også lurt å legge til en instruksjon til hva som skjer dersom if-setningen ikke er sann. I vårt eksempel blir dette når spilleren har mer enn 0 «health», altså når spilleren fremdeles er i live. Dette gjøres med en «else-setning» og skrives på følgende måte:

// Dersom variablen «gameOver» er usann skrives det ut en passende melding.

else {

Debug.Log («You're still alive buddy!»);

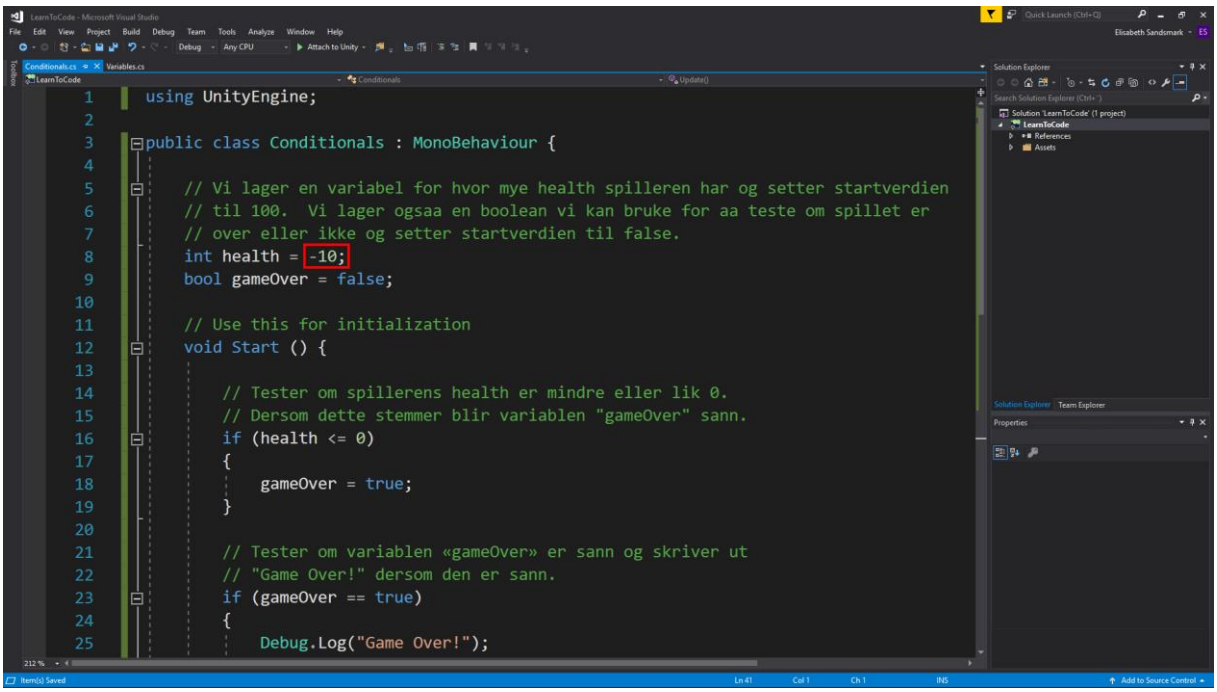
}



```
        // Tester om spillerens health er mindre eller lik 0.
        // Dersom dette stemmer blir variablen "gameOver" sann.
        if (health <= 0)
        {
            gameOver = true;
        }

        // Tester om variablen «gameOver» er sann og skriver ut
        // "Game Over!" dersom den er sann.
        if (gameOver == true)
        {
            Debug.Log("Game Over!");
        }

        // Dersom variablen "gameOver" er usann skrives det ut en passende melding.
        else
        {
            Debug.Log("You're still alive buddy!");
        }
    }

    // Update is called once per frame
    void Update () {
```

Hvis du tester koden din i Unity nå (OBS! Husk alltid å lagre før du tester koden din) vil du få meldingen at du fremdeles er i live siden startverdien til health er satt til 100. Hvis du endrer health til f.eks. -10 vil du få beskjeden om at spillet er over.

int health = -10;



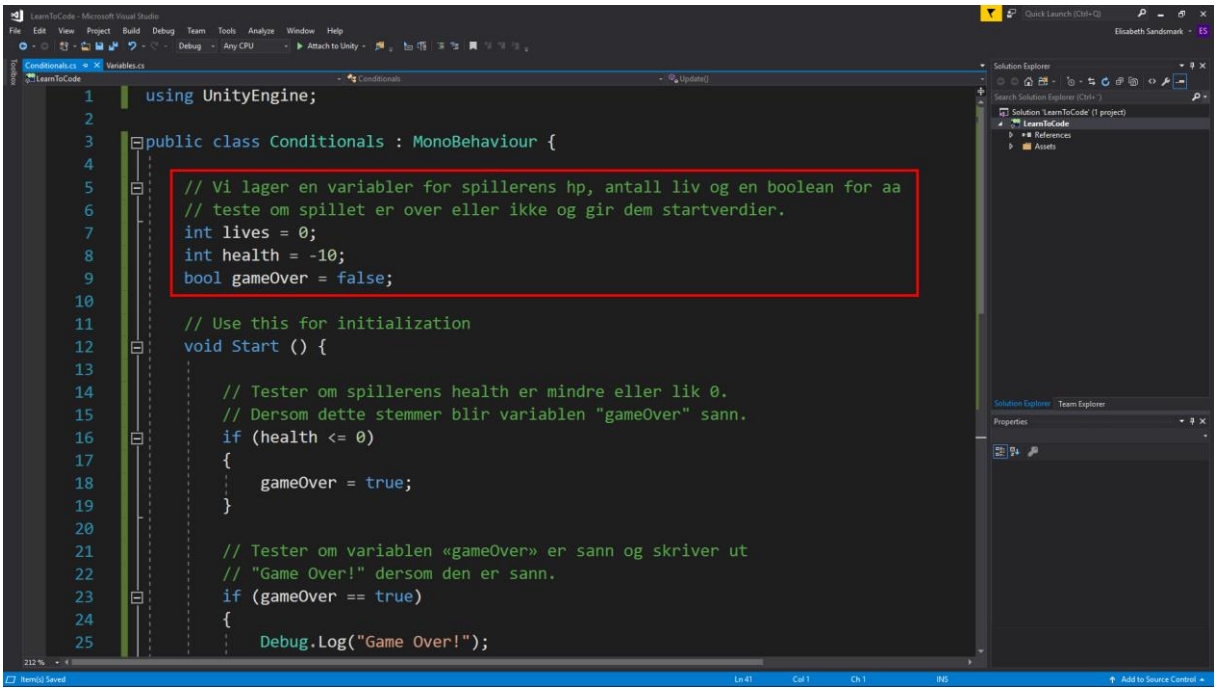
```
using UnityEngine;

public class Conditionals : MonoBehaviour {

    // Vi lager en variabel for hvor mye health spilleren har og setter startverdien
    // til 100.  Vi lager ogsaa en boolean vi kan bruke for aa teste om spillet er
    // over eller ikke og setter startverdien til false.
    int health = -10;
    bool gameOver = false;

    // Use this for initialization
    void Start () {

        // Tester om spillerens health er mindre eller lik 0.
        // Dersom dette stemmer blir variablen "gameOver" sann.
        if (health <= 0)
        {
            gameOver = true;
        }

        // Tester om variablen «gameOver» er sann og skriver ut
        // "Game Over!" dersom den er sann.
        if (gameOver == true)
        {
            Debug.Log("Game Over!");
```

Eksempel 2

I dette eksemplet ønsker vi at spilleren skal kunne ha mer enn ett liv, f.eks. 3. Etter at alle 3 livene er borte og spilleren har 0 health igjen vil spilleren få en melding om at spillet er over.

Vi begynner med å lage en variabel for antall liv og setter startverdien til 0:

int lives = 0;

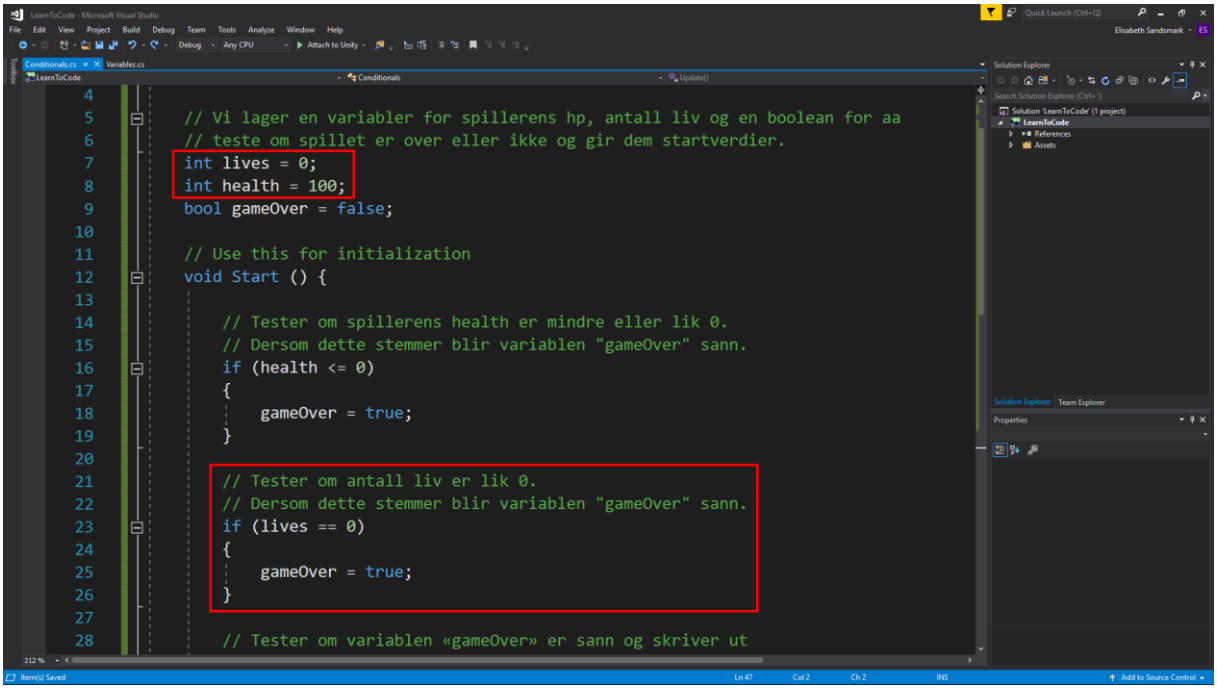


Vi ønsker nå at spillet skal være over når spillerens health er mindre eller lik 0 **og/samtidig som** spilleren har 0 liv igjen. For å gjøre dette kan vi ikke skrive en ny if-setning på samme måte som vi har gjort tidligere. Vi trenger at begge uttrykkene er sanne samtidig.

Dersom du lager en ny if-setning for «lives» på samme måte som du gjorde for «health» vil ikke verdien for «gameOver» nødvendigvis bli riktig. Tenk deg for eksempel at spilleren har 100 health igjen og er på sitt siste liv (0 ekstra liv). Skriver vi koden på følgende måte vil «gameOver» først få verdien «true», deretter vil den få verdien «false» (husk at koden leses fra linje 1 og nedover), og det vil skrives ut «Game Over!» selv om spilleren fremdeles har 100 health igjen.

OBS! Du trenger ikke skrive inn dette i koden, det er bare for å vise hvordan det ville sett ut dersom du lager en egen if-setning for «lives»!

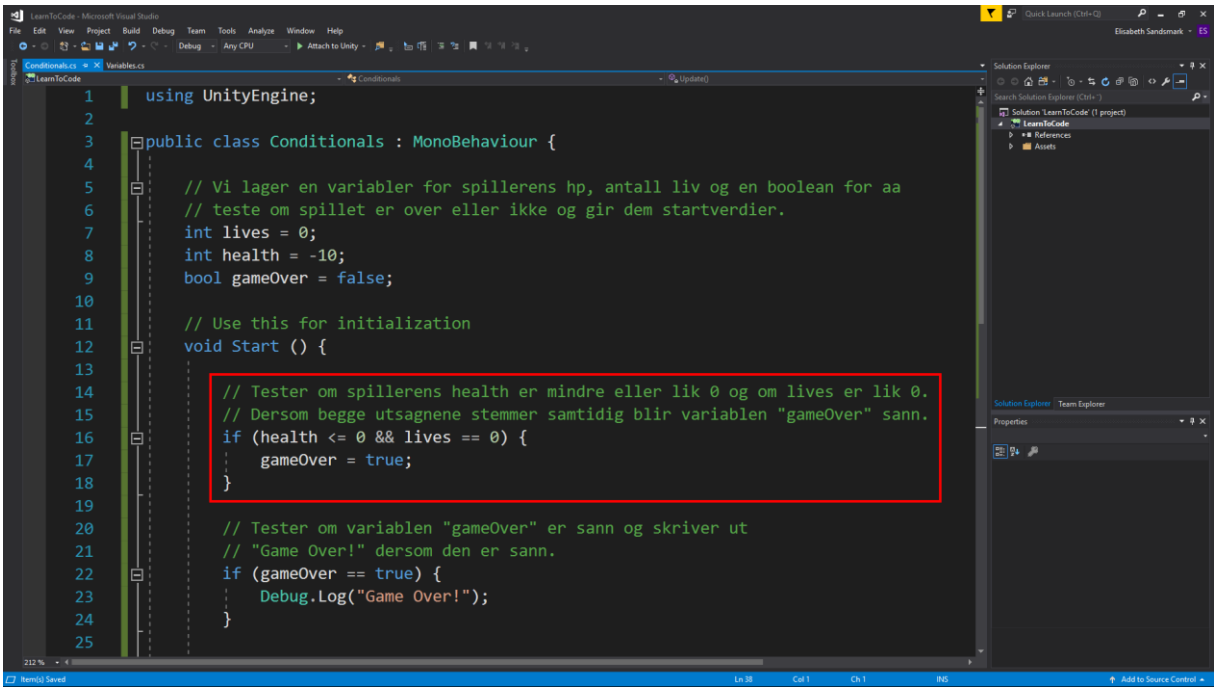
```
int lives = 0;

int health = 0;


if (health <= 0) {
        gameOver = true;
}

if (lives == 0) {
        gameOver = true;
}
```



Vi må derfor heller skrive følgende:

// Tester om spillerens health er mindre eller lik 0 og om lives er lik 0. Dersom begge utsagnene stemmer samtidig blir variablen «gameOver» sann.

```
if (health <= 0 && lives == 0) {
        gameOver = true;
}
```



NB! Vi kan bruke flere ulike tegn mellom to eller flere utsagn i samme if-setning for å teste og utføre instruksjoner til flere utsagn samtidig. && står for «og». || står for «eller». Dersom du bruker && må begge utsagnene være sanne, men dersom du bruker || behøver kun ett av utsagnene å være sant for at instruksjonen skal utføres.

Eksempel 3

Tenk deg at du ønsker å lage et spill i likhet med «Clash Royale». Et spill hvor du har en base bestående av to tårn og et hovedtårn. Målet med spillet er å ødelegge fiendens hovedtårn.

For å kunne avgjøre hvem som vinner må vi først velge hva som avgjør seier/tap

- Dersom hovedtårnet til en av spillerne blir ødelagt
- Dersom tiden går ut

Det er også mulig at spillet ender i en «draw» (det blir uavgjort).

OBS! Du trenger ikke lage et nytt skript til dette eksempelet. Du kan enten stryke ut koden du allerede har eller legge det til den nye koden under koden du allerede har skrevet.

Vi begynner med å lage noen variabler. Blant annet trenger vi variabler for hvor mange tårn som er igjen, og vi setter startverdien til 2 siden hver spiller har 2 tårn utenom hovedtårnet.

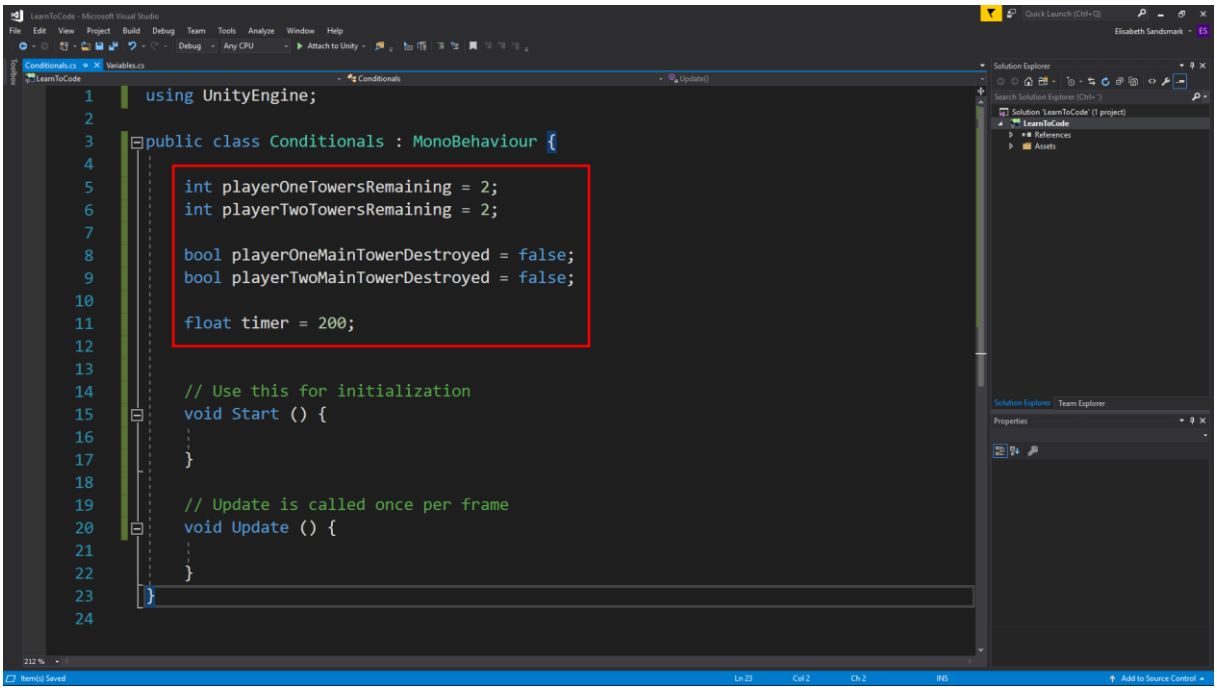
```
int playerOneTowersRemaining = 2;
int playerTwoTowersRemaining = 2;
```

Vi trenger også en boolean for å teste om hovedtårnene til de to spillerne har blitt ødelagt. Vi setter startverdien til false siden tårnene ikke er ødelagte i starten.

```
bool playerOneMainTowerDestroyed = false;
bool playerTwoMainTowerDestroyed = false;
```

Vi trenger også en variabel for timeren for å vite hvor mye tid som gjenstår av kampen. Sett startverdien til f.eks. 200 (sekunder).

```
float timer = 200;
```



Det første vi ønsker å teste er om et av hovedtårnene har blitt ødelagt. Vi lager en if-setning for å teste dette. Husk at || betyr «or/eller». Dersom hovedtårnet til spiller 1 eller spiller 2 er ødelagt så skal instruksjonene innenfor if-setningen utføres.

```
if (playerOneMainTowerDestroyed || playerTwoMainTowerDestroyed) {
}
```

Vi kan deretter teste om det er hovedtårnet til spiller 1 eller om det er hovedtårnet til spiller 2 som er ødelagt.

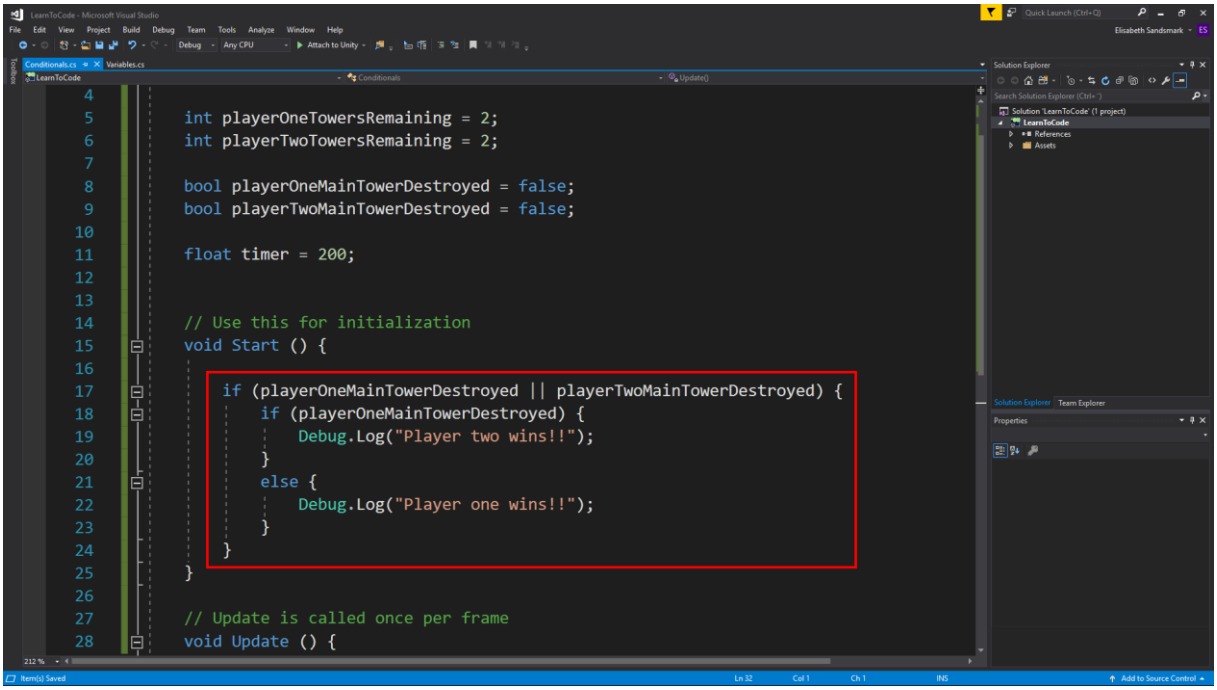
Først lager vi en if-setning for å teste den første påstanden «spiller 1 sitt tårn er ødelagt» er sann eller usann. Deretter lager vi noe som heter «else-setning». Denne vil kjøre dersom if-setningen ikke er sann (i dette tilfellet når spiller 2 sitt tårn er ødelagt).

```
if (playerOneMainTowerDestroyed || playerTwoMainTowerDestroyed) {
        if (playerOneMainTowerDestroyed) {
        } else {
        }
}
```

Dersom hovedtårnet til spiller 1 er ødelagt vil det skrives ut at spiller 2 er vinneren. Dersom hovedtårnet til spiller 2 er ødelagt vil det skrives ut at spiller 1 er vinneren.

```
if (playerOneMainTowerDestroyed || playerTwoMainTowerDestroyed) {
        if (playerOneMainTowerDestroyed) {
                Debug.Log («Player two wins!!»);
        } else {
                Debug.Log(«Player one wins!!»);
        }
}
```

Du kan fint legge til så mange if-setninger du ønsker inni andre if-setninger, og det er veldig vanlig å gjøre når man lager spill!



Det neste vi skal teste er om tiden har gått ut. Vi lager en «else if-setning» for å sjekke om tiden er mindre enn eller lik 0 og kjører instruksene dersom det er sant.

En else if-setning er en kombinasjon av en if-setning og en else-setning. Den brukes for å utføre instrukser dersom den originale if-setningen ikke er sann samtidig som betingelsene for else if-setningen er sann. Else if-setninger skrives etter if-setningen og før else-setningen. OBS! Det er alltid kun én else-setning for hver if-setning, men du kan ha så mange else if-setninger som du måtte ønske!

```
else if (timer <= 0) {
}
```

Først tester vi om spiller 1 har færre tårn etter at tiden har gått ut enn spiller 2 med en if-setning. Dersom spiller 1 har færre tårn er spiller 2 vinneren og vi skriver ut at spiller 2 vant.

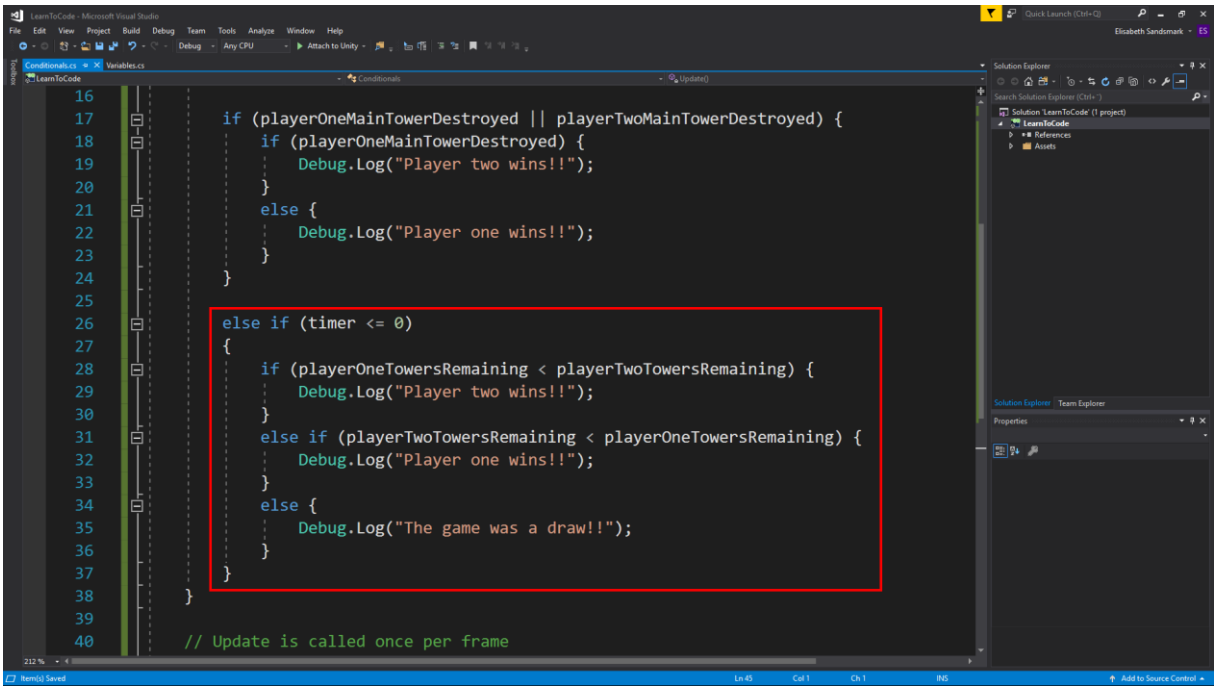
```
else if (timer <= 0) {
    if (playerOneTowersRemaining < playerTwoTowersRemaining) {
        Debug.Log («Player two wins!!»);
}
```

Deretter tester vi om spiller 2 har færre tårn enn spiller 1 med en else if-setning. Dersom dette er sant er spiler 1 vinneren og resultatet skrives ut.

```
else if (timer <= 0) {
    if (playerOneTowersRemaining < playerTwoTowersRemaining) {
        Debug.Log («Player two wins!!»);
    } else if (playerTwoTowersRemaining < playerOneTowersRemaining) {
        Debug.Log(«Player one wins!!»);
}
```

Det kan jo også hende at begge spillerne har like mange tårn igjen når tiden har gått ut og spillet ender med uavgjort. Vi må derfor lage en else-setning for å lage instrukser for dette også. (else-setningen kjøres dersom if- og else if-setningene ikke er sanne, altså når begge spillerne har like mange tårn igjen i dette tilfellet.)

```
else if (timer <= 0) {
    if (playerOneTowersRemaining < playerTwoTowersRemaining) {
        Debug.Log («Player two wins!!»);
    } else if (playerTwoTowersRemaining < playerOneTowersRemaining) {
        Debug.Log(«Player one wins!!»);
    } else {
        Debug.Log(«The game was a draw!!»);
    }
}
```



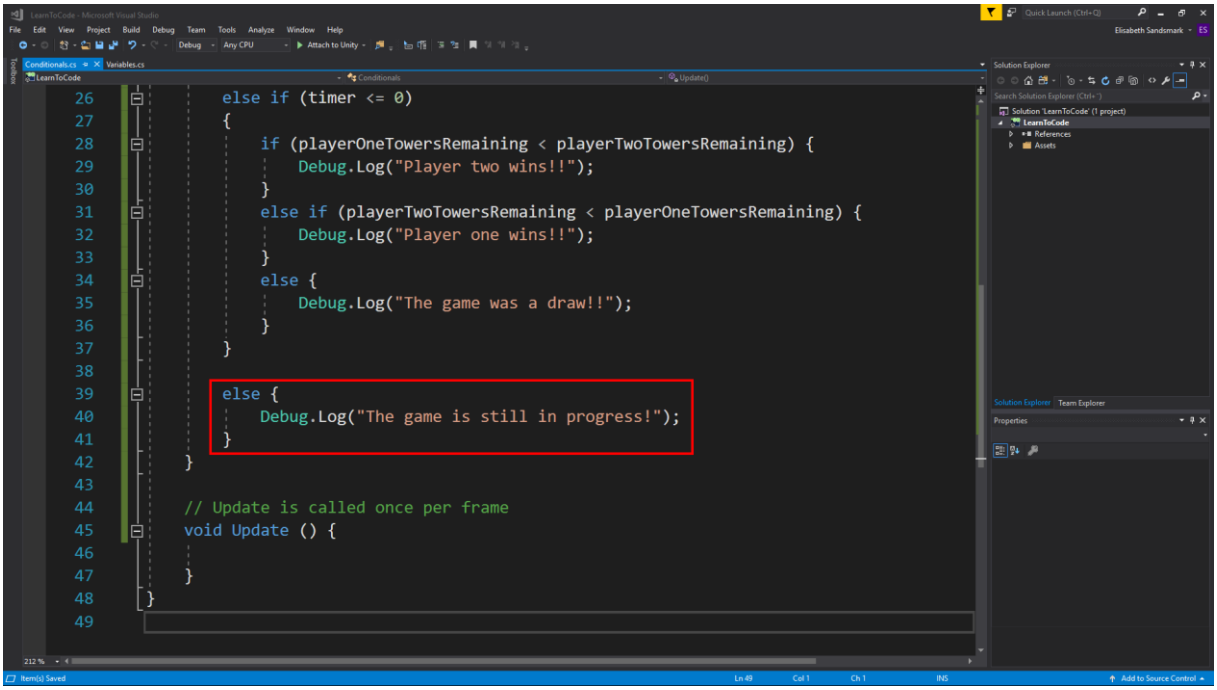
```
16
17          if (playerOneMainTowerDestroyed || playerTwoMainTowerDestroyed) {
18              if (playerOneMainTowerDestroyed) {
19                  Debug.Log("Player two wins!!");
20              }
21              else {
22                  Debug.Log("Player one wins!!");
23              }
24          }
25
26          else if (timer <= 0)
27          {
28              if (playerOneTowersRemaining < playerTwoTowersRemaining) {
29                  Debug.Log("Player two wins!!");
30              }
31              else if (playerTwoTowersRemaining < playerOneTowersRemaining) {
32                  Debug.Log("Player one wins!!");
33              }
34              else {
35                  Debug.Log("The game was a draw!!");
36              }
37          }
38      }
39
40      // Update is called once per frame
```

Til slutt legger vi til en else-setning som brukes dersom spillet fremdeles ikke har noen vinner.

else {

Debug.Log(«The game is still in progress!»)

}



```
26          else if (timer <= 0)
27          {
28              if (playerOneTowersRemaining < playerTwoTowersRemaining) {
29                  Debug.Log("Player two wins!!");
30              }
31              else if (playerTwoTowersRemaining < playerOneTowersRemaining) {
32                  Debug.Log("Player one wins!!");
33              }
34              else {
35                  Debug.Log("The game was a draw!!");
36              }
37          }
38
39          else {
40              Debug.Log("The game is still in progress!");
41          }
42      }
43
44      // Update is called once per frame
45      void Update () {
46
47      }
48  }
49
```

Du kan f.eks. teste hva som skjer dersom du endrer verdien til timer til 0 eller endrer verdien til en boolean til true.

Oppgave

Se på følgende kode. Finn ut hva som kommer til å skje og hvorfor uten å kjøre koden! Kommer den til å kjøre?

```
if (true == false || false != true && 1 == 1) {
        Debug.Log («Did we get here»);
}
```

Løsning:

La oss se på første del av setningen «**true == false**» sant/true er lik usant/false. Vi vet at resultatet av dette vil bli **false**. Du kan ikke ha noe som er sant og usant samtidig!

Heldigvis har vi en **||** (eller/or) som gjør at utsagnet fremdeles kan være sant dersom det som utsagn nr. to er sant.

Utsagn nummer to, «**false != true**» usant/false er ikke lik sant/true (det som er usant er ikke lik det som er sant). Dette er **sant**.

Dermed har vi en **&&** (og/and) som gjør at utsagnet på venstre side og utsagnet på høyre side begge må være sanne samtidig.

Utsagn nummer tre, «**1 == 1**» én er lik én er **sant**.

Vi kan se på det på følgende:

(true == false || false != true) && 1 == 1

(usant eller sant) og sant

sant og sant

sant!

Vi vet dermed at logikken stemmer og at programmet fint vil kunne kjøre.

Kort oppsummert

En boolean er en logisk variabel som består av operatorer (Eksempler: «og», «eller», «hverken eller») og har en verdi som er enten true/sann eller false/usann.

Relasjonsoperatorer (f.eks. ==, >=, !=) brukes for å avgjøre om en boolean er true/sann eller false/usann.

If-setninger brukes for å teste om en betingelse er sann. Dersom betingelsen er sann blir en instruksjon utført. Else-setninger brukes for å utføre en instruksjon dersom betingelsene i if-setningen ikke er sanne. Else if-setninger er en kombinasjon av if og else og brukes til å utføre instruksjoner dersom den originale if-setningen ikke er sann samtidig som betingelsene for else if-setningen er sann.

Logikk og if-setninger er svært viktig når du koder spill! Tenk bare over hvor mange valgmuligheter som ligger bak spillene du spiller, og hvor viktig det er at logikken stemmer. «Hvis jeg trykker på denne knappen så skjer dette.» «Hvis jeg plukker opp denne gullmynten har jeg 100 gullmynter til sammen og hvis jeg har 100 gullmynter kan jeg kjøpe det sverdet men ikke det andre sverdet som koster 200. Sverdet blir også billigere hvis jeg har gjort et quest for personen som selger det…»