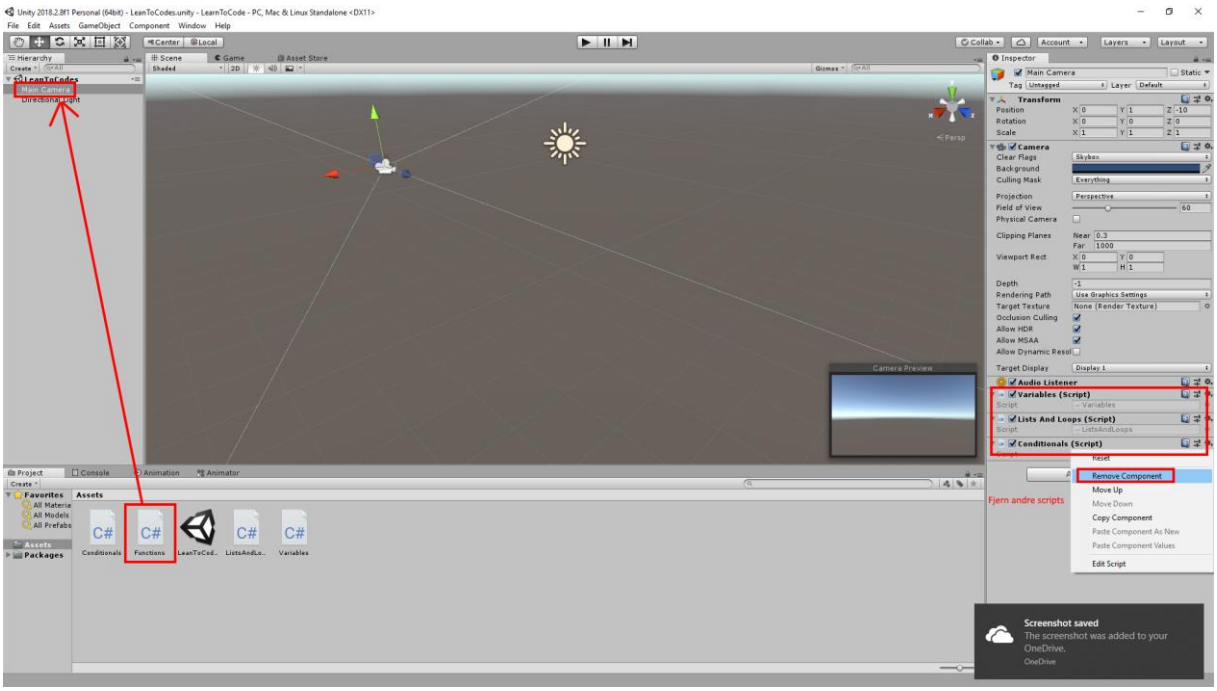
Lær å kode i Unity – del 4: Funksjoner/metoder

I denne delen skal vi lære å jobbe med funksjoner/metoder.

Lag et nytt C# script og kall det «Functions».

Husk å legge til det nye scriptet under Main Camera! (Du kan også fjerne de gamle scriptene fra Main Camera.



En funksjon er en måte å oppbevare en serie med instruksjoner. Store programmer inneholder ofte mange funksjoner og er fundamentet av programmet. Funksjoner kan blant annet manipulere variabler og kalle på andre funksjoner.

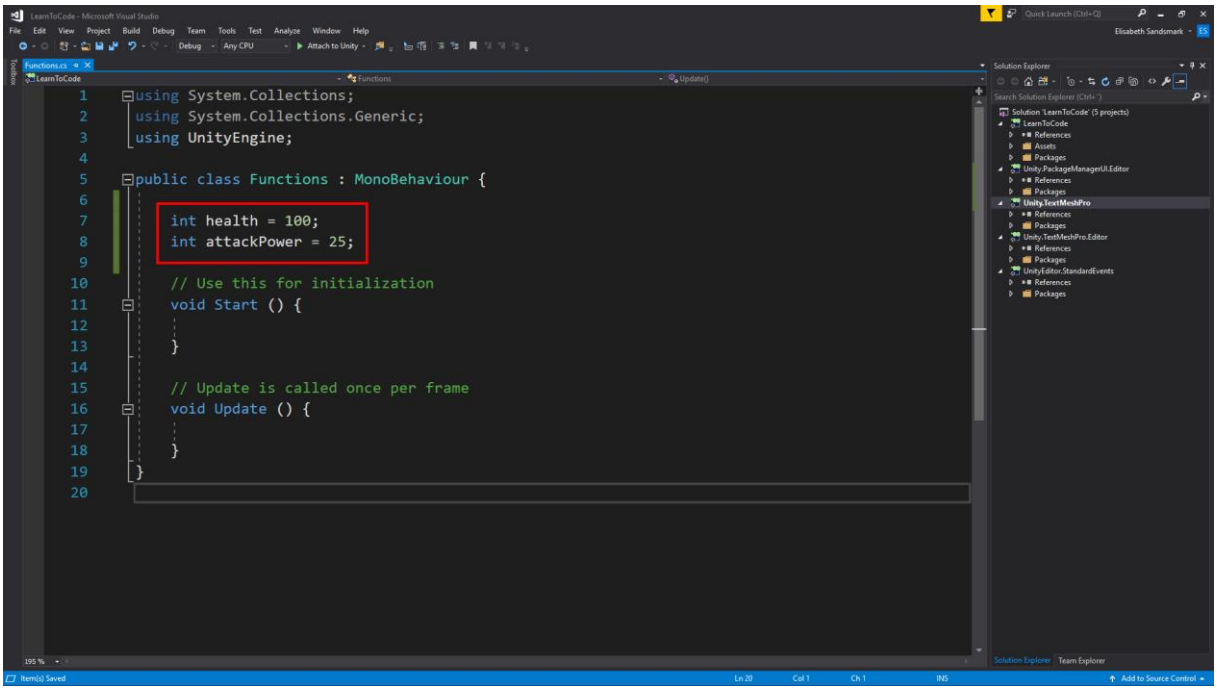
En funksjon kan enten returnere noe eller ikke returnere noe, den har et navn (for eksempel «Start») og den kan ha en eller flere parametere (det som står innenfor parentesene).

Til nå har vi bare sett to funksjoner «void Start ()» og «void Update ()». «Start» funksjonen er en funksjon som ligger inne i Unity og kalles på når spillet startet (en gang). Det samme er «Update» funksjonen, men denne kalles på hver gang spillet oppdaterer seg (x ganger i sekundet, for eksempel 60 ganger («frames») i sekundet). Dette er bare to eksempler på funksjoner og vi skal nå prøve å lage våre egne funksjoner.

Vi later som om vi har et battle arena spill hvor spillerne skal kjempe mot hverandre. Vi lager noen variabler som vi kan bruke i spillet vårt. Vi tenker oss også at vi har en knapp. Hver gang vi klikke på knappen mister motstanderen litt health.

int health = 100;

int attackPower = 25;



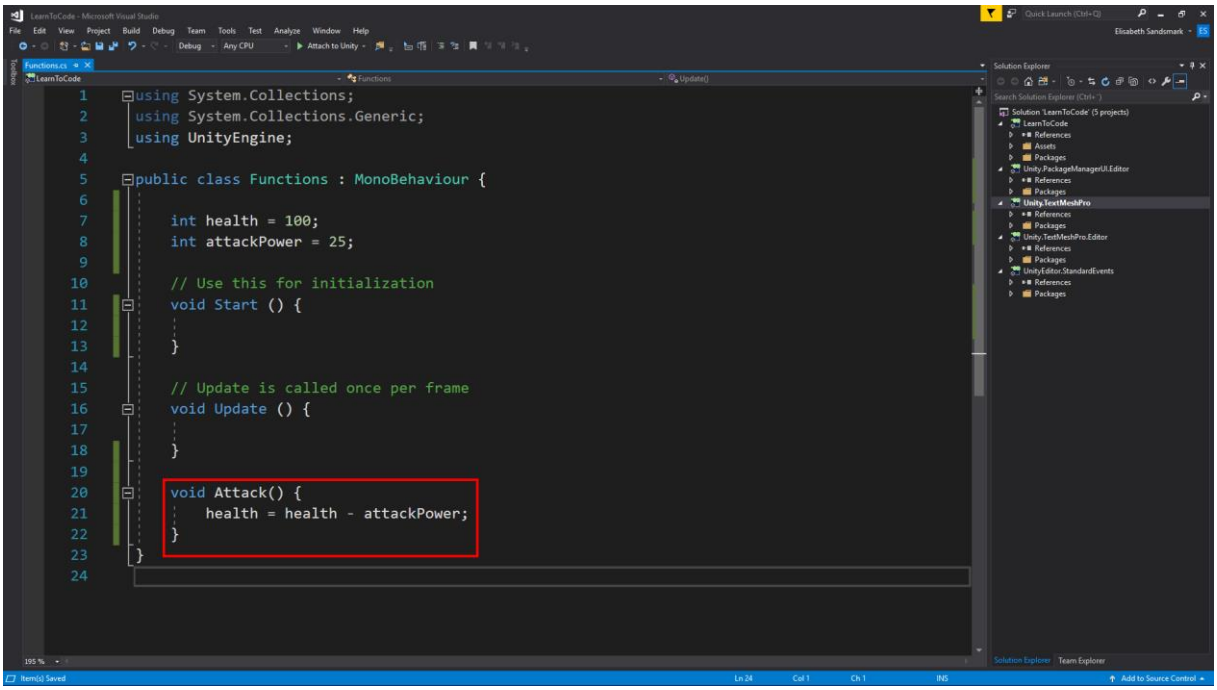
Vi lager vår første funksjon. Vi ønsker at hver gang Attack funksjonen kalles på så utføres en operasjon som trekker 25 health fra spilleren og lagre den nye verdien for health i health variablen.

Void vil si at det ikke returneres noe i funksjonen.

Navnet på funksjonen er «Attack». (Et funksjonsnavn må ikke starte med stor bokstav, men det er normalt å gjøre det i C#.)

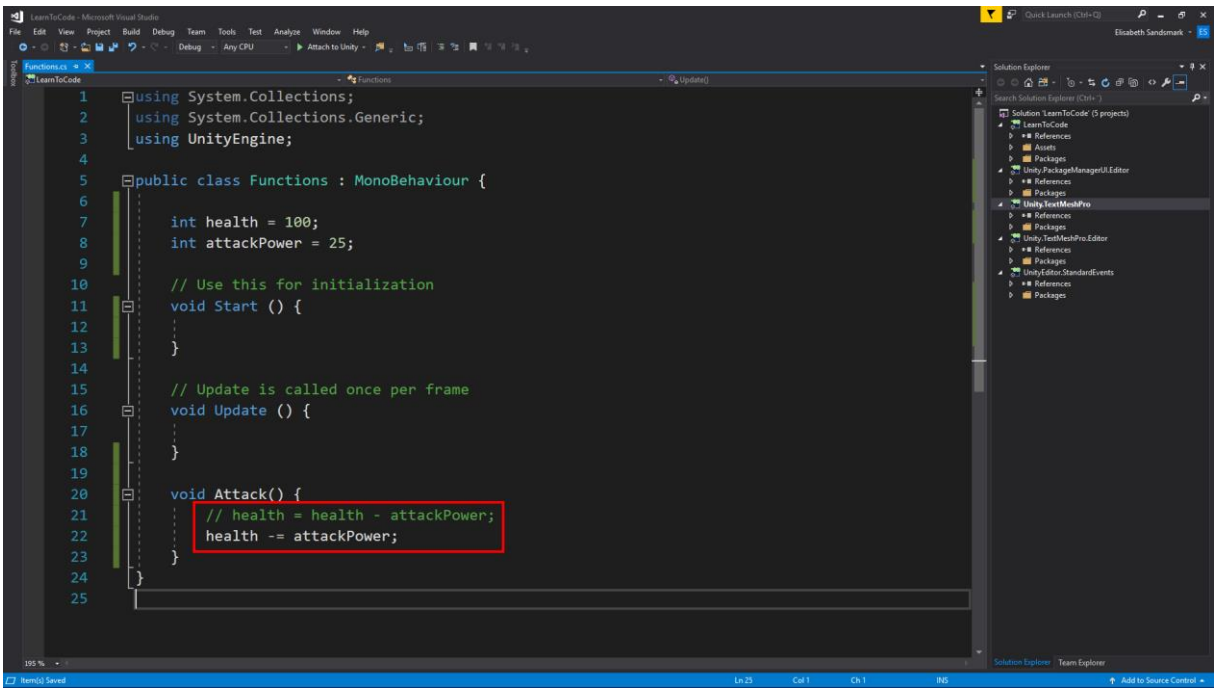
Det er ingen parametere i parentesene.

```
void Attack () {
        health = health – attackPower;
}
```



En annen måte å skrive dette på er som følger (denne funskjonen gjør nøyaktig det samme som den første):

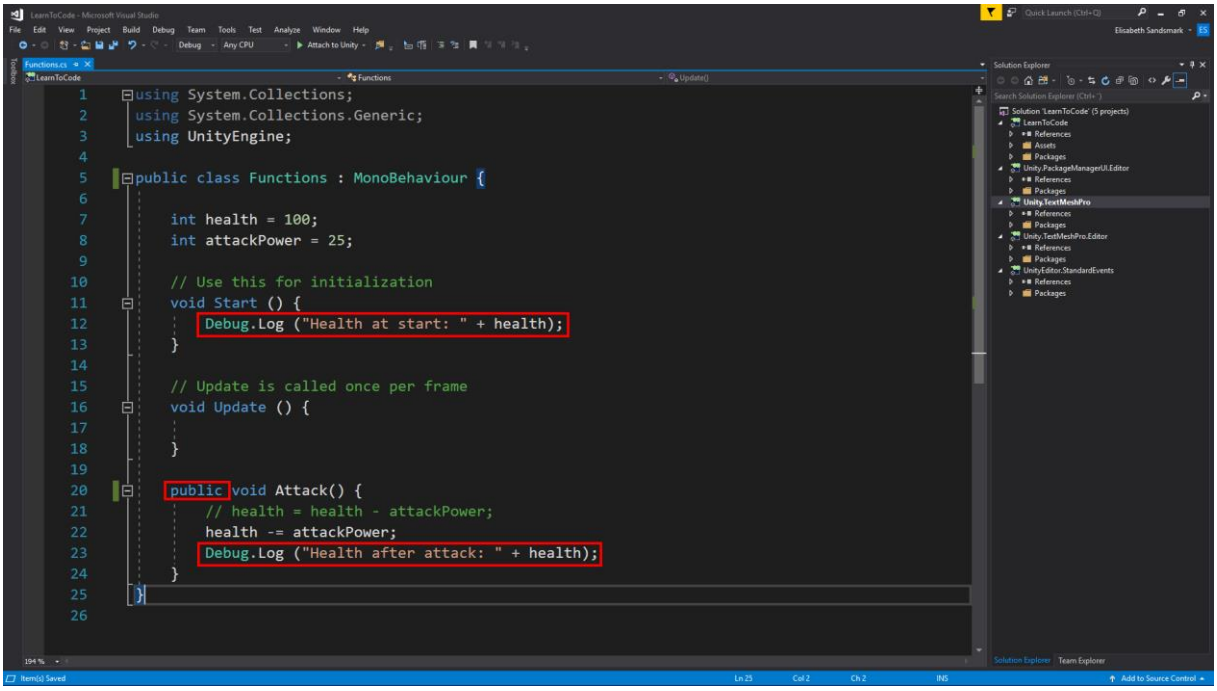
```
void Attack () {
        health -= attackpower;
}
```



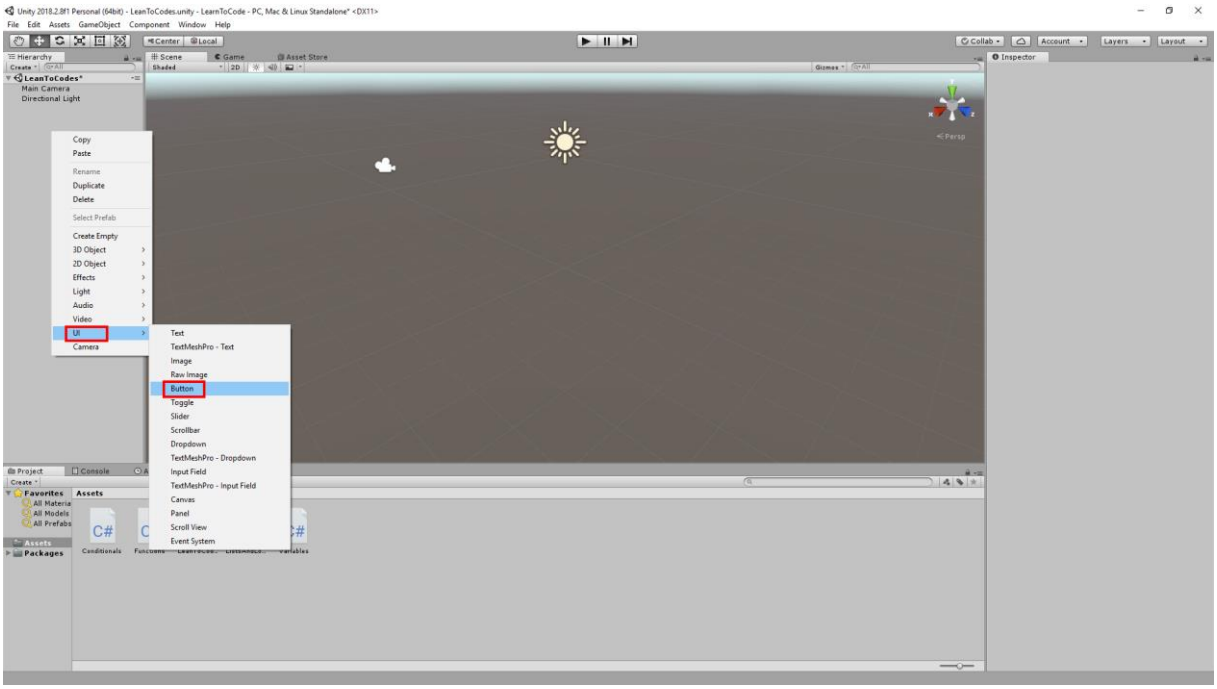
En funksjon kan enten være private eller public. Vi skal se nærmere på dette i neste del, men kort fortalt vil private si at funksjonen kun kan brukes i den «klassen» (Ikke andre C# script) og public vil si at funksjonen kan hentes i andre «klasser» og i Unity.

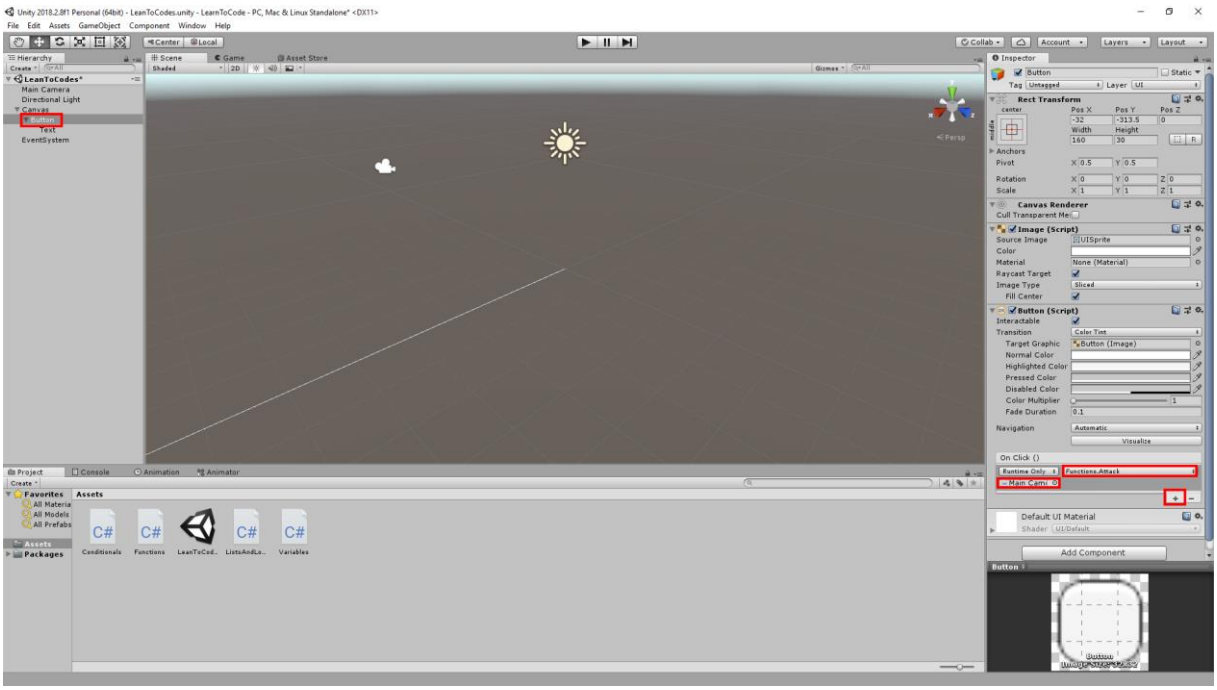
I dette tilfellet ønsker vi at funksjonen skal være public. Skriv ut hvor mye health spilleren har igjen etter et attack til Unity. Skriv også ut hvor mye health spilleren har i starten under Start funksjonen.

```
void Start () {
        Debug.Log(«health at start: » + health);
}
public void Attack () {
        health -= attackpower;
        Debug.Log(«Health after attack: » + health);
}
```

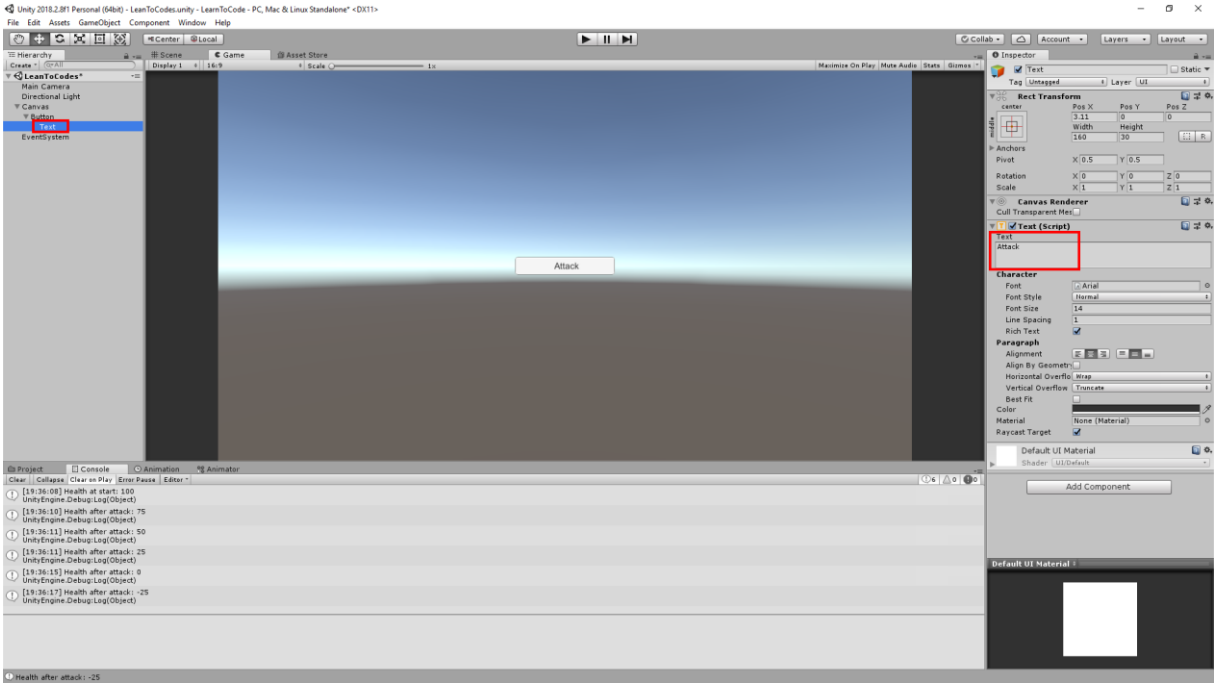


For å teste at spilleren mister health hver gang vi bruker attack knappen må vi lage en knapp vi kan bruke. Dette gjør vi i Unity. Høyreklikk under Hierarchy og klikk UI – Button. Dette gjør at det popper opp en knapp midt på skjermen. Vi ønsker at knappen skal kalle på attack funksjonen. Dette gjør vi med å klikke på knappen under Hierarchy, klikk på pluss tegnet der det står «On Click ()» og dra deretter «Main Camera» til der det står «None». Endre deretter «No Function» til «Functions» - «Attack ()». Dersom du ikke hadde hatt funksjonen public hadde den ikke blitt vist her.

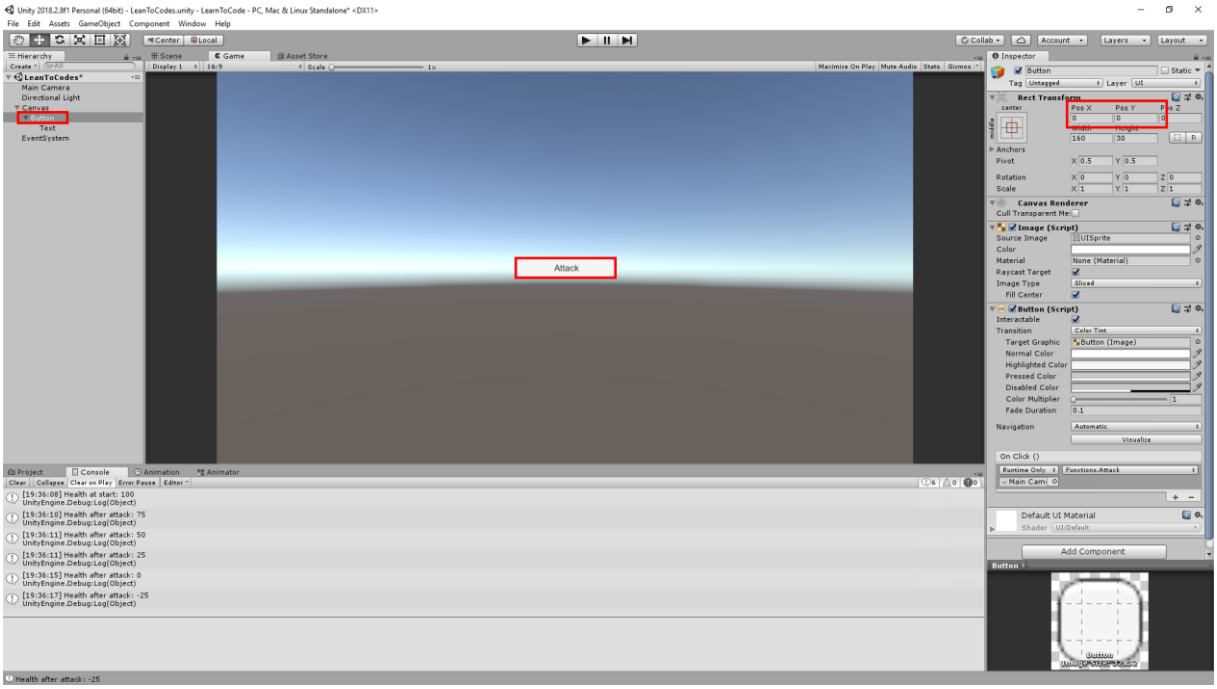




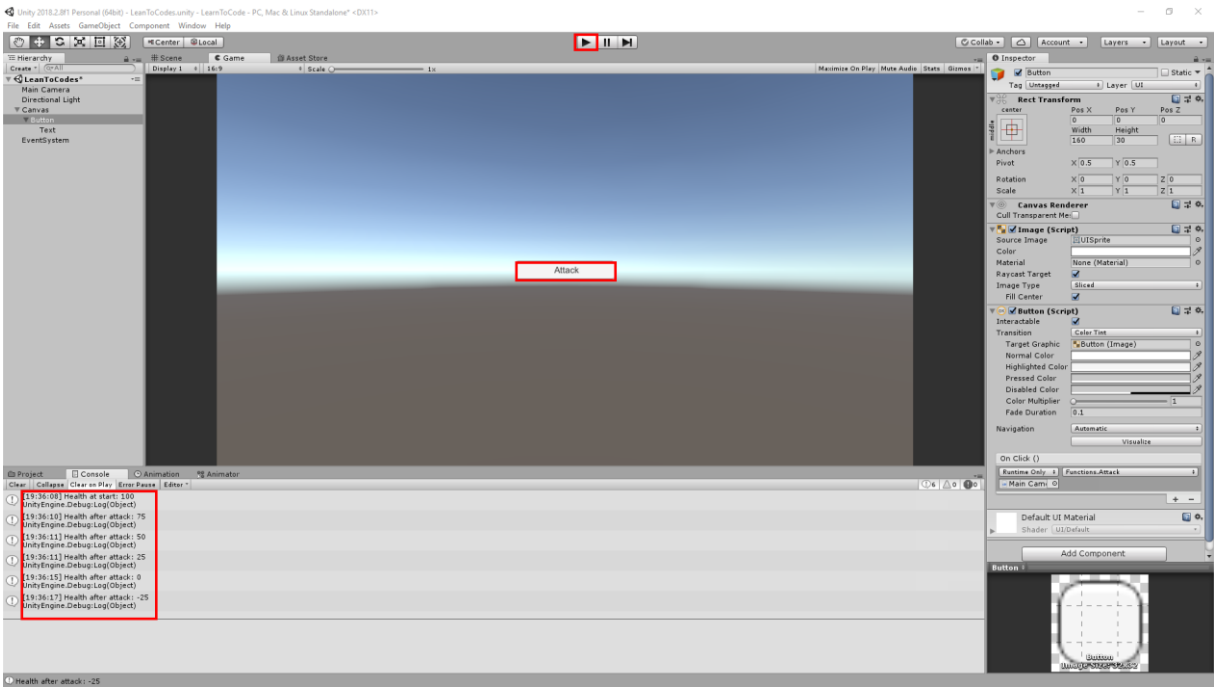
(Du kan endre navnet på knappen til for eksempel attack under «Text»)



Klikk på «Game» vinduet for å sjekke om knappen er der. Dersom den ikke er der må du endre posisjonen til 0 på X og Y til 0.



Lagre alt og klikk play for å teste at det fungerer. Hver gang du trykker på attack knappen skal du miste 25 health.



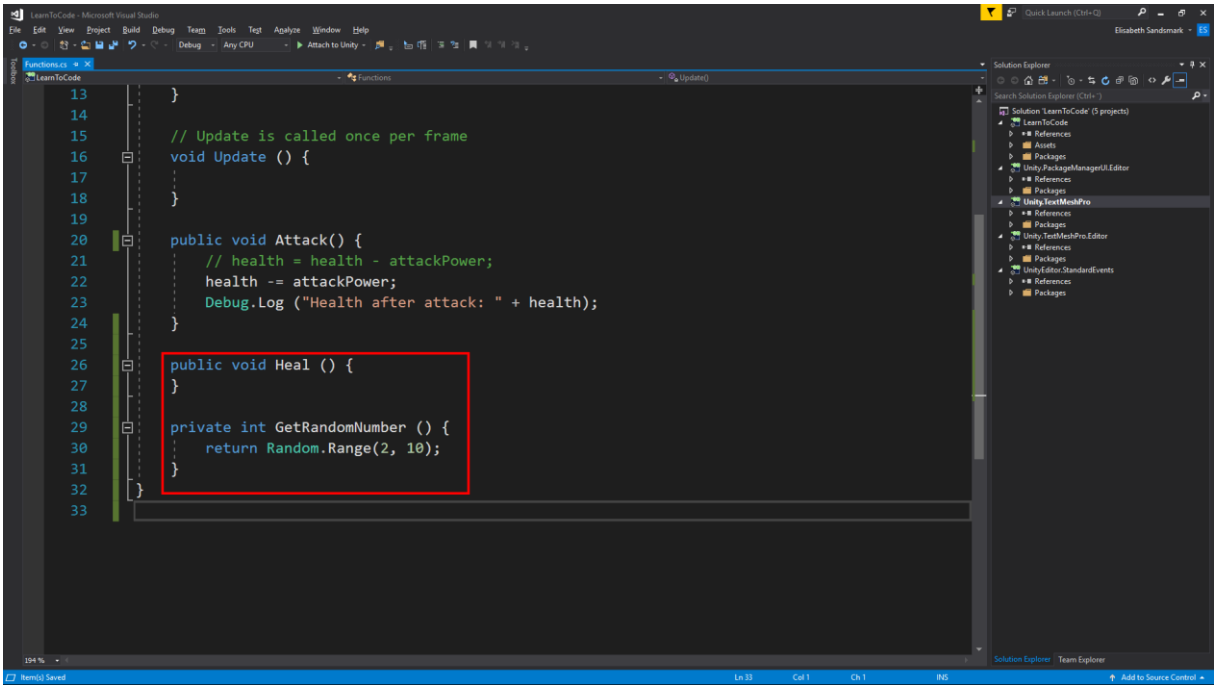
<u>Kort oppsummering:</u> Det vi nå har gjort er å kalle på en funksjon. Funksjonen har utført en operasjon og printet ut det den skulle.

Tenk deg nå at spilleren går over en mystisk boks. Denne boksen har en sjanse for å gi spilleren litt health tilbake, men du vet ikke hvor mye den gir. Den kan gi litt health eller mye health. La oss opprette en ny funksjon for dette.

```
public void Heal () {

}
```

Vi lager også en private funksjon for å returnere en iteger (int) for et tilfeldig tall fra 2 og opp til 10.

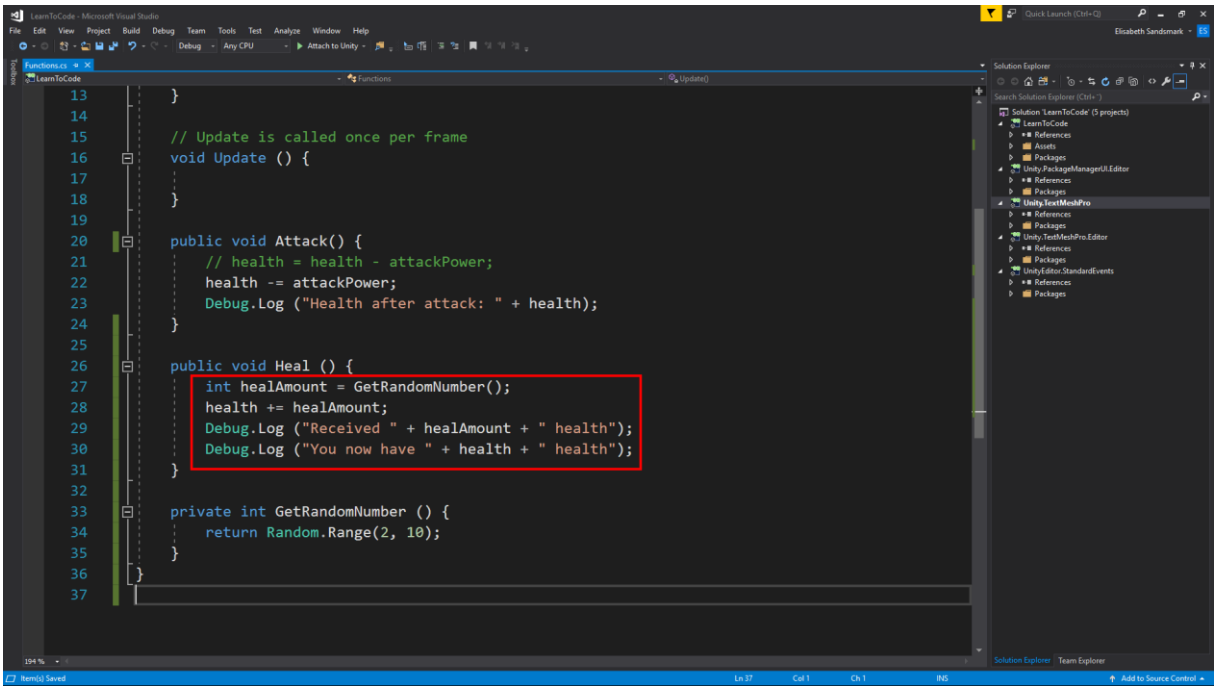
```
private int GetRandomNumber () {

        return Random.Range (2, 10);

}
```



Vi skal lage en knapp for å kalle på Heal funksjonen, men først må vi kalle på funksjonen GetRandomNumber i Heal funksjon.

Vi oppretter en variabel for hvor mye vi healer slik at vi kan skrive ut både hvor mye vi healer og hvor mye health vi har.

```
public void Heal () {

        int healAmount = GetRandomNumber ();

        health += healAmount;

        Debug.Log («Received » + healAmount + « health»);

        Debug.Log(«You now have » + health + « health»);

}
```

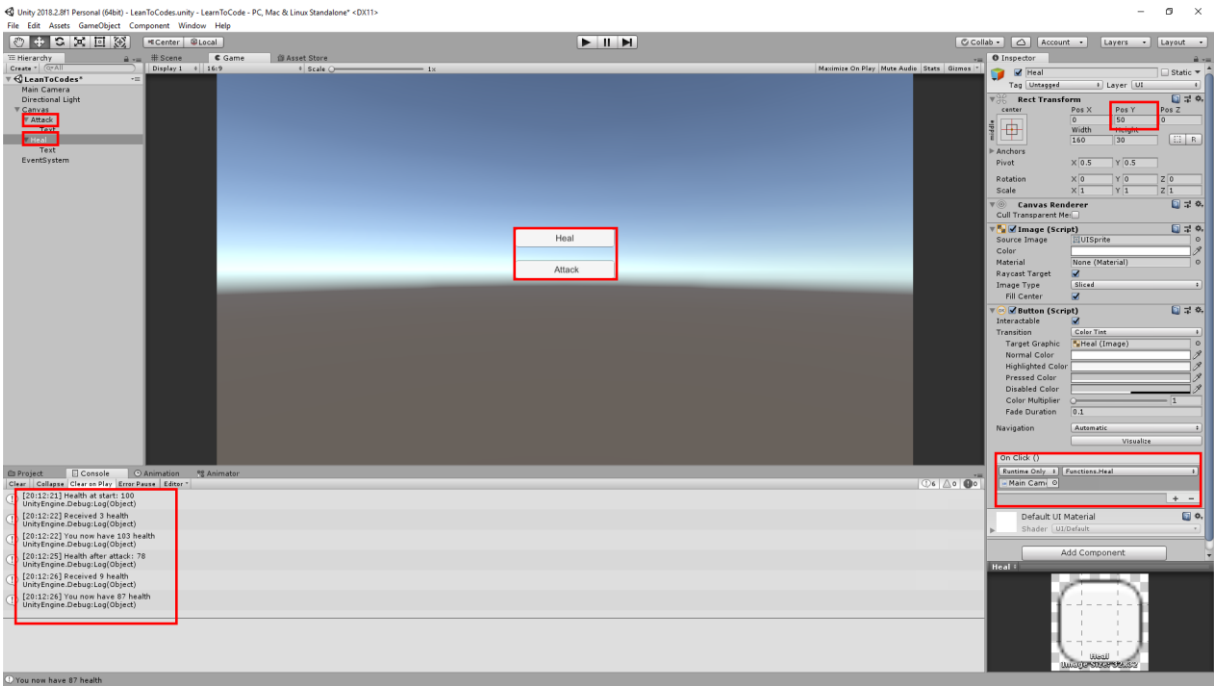


```
    }

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

    }

    public void Attack() {
        // health = health - attackPower;
        health -= attackPower;
        Debug.Log ("Health after attack: " + health);
    }

    public void Heal () {
        int healAmount = GetRandomNumber();
        health += healAmount;
        Debug.Log ("Received " + healAmount + " health");
        Debug.Log ("You now have " + health + " health");
    }

    private int GetRandomNumber () {
        return Random.Range(2, 10);
    }
}
```

Lag en ny knapp slik som du gjorde med attack knappen. Endre Y posisjonen til for eksempel 50 (slik at du kan se begge knappene). Legg til Main Camera under OnClick og endre fra No Function til Functions – Heal ().

Du kan også endre navnet på knappene til Attack og Heal for å få litt bedre oversikt.

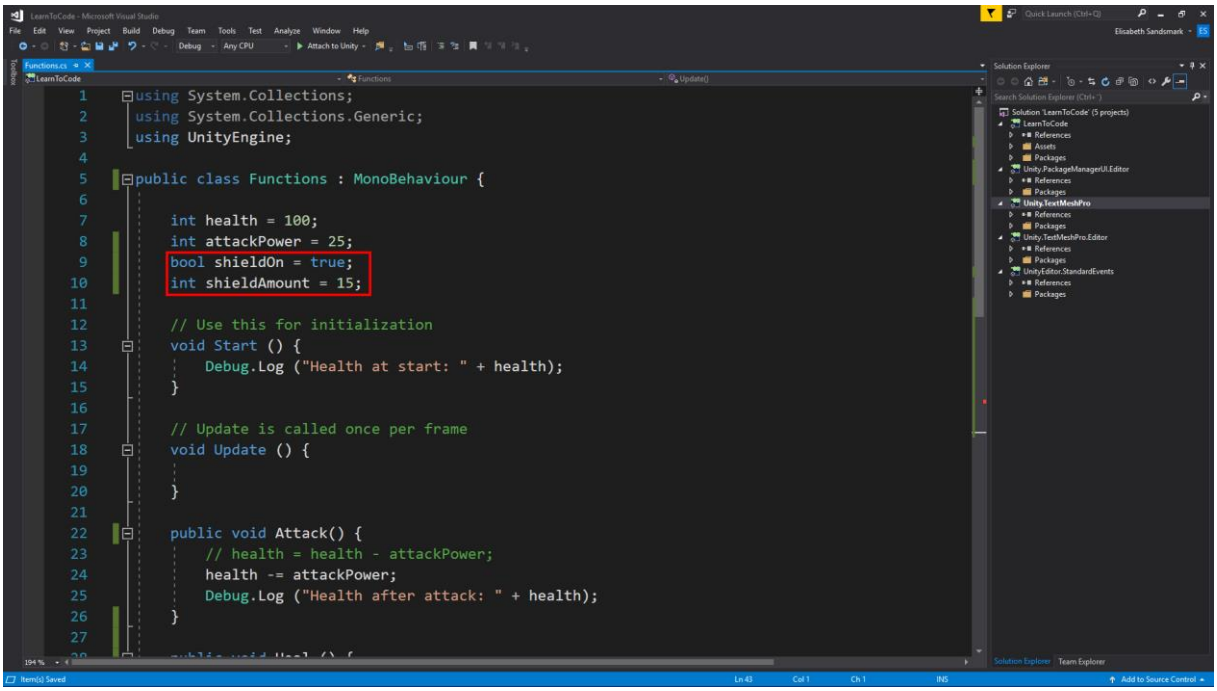
Test og sjekk at alt fungerer.



Tenk deg nå at spilleren har et shield. Vi lager først en bool (kan enten være true eller false) for å sjekke om spilleren har shield eller ikke. Vi lager også en int for å si hvor mye shield spilleren har.

bool shieldOn = true;

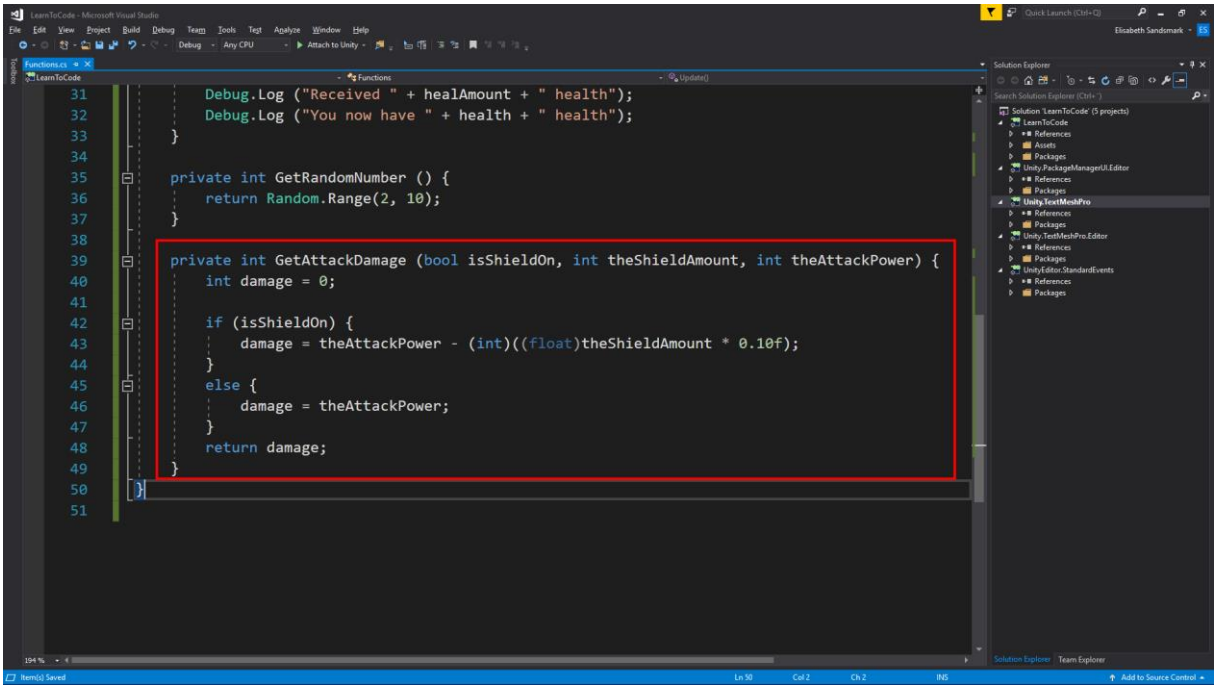
int shieldAmount = 15;



Det holder ikke lenger å bare bruke attackPower. Vi lager en ny funksjon for å kalkulere attack damage. Vi ønsker å se om vi har et aktivt shield, og dersom vi har det skal det først fjerne fra shieldet før health. Vi lager derfor tre parametere (det som er inne i parentesene). Når du kaller en funksjon med parametere så overfører du data til plasser i funksjonen og gir dem nye navn slik at funksjonen kan bruke dem.

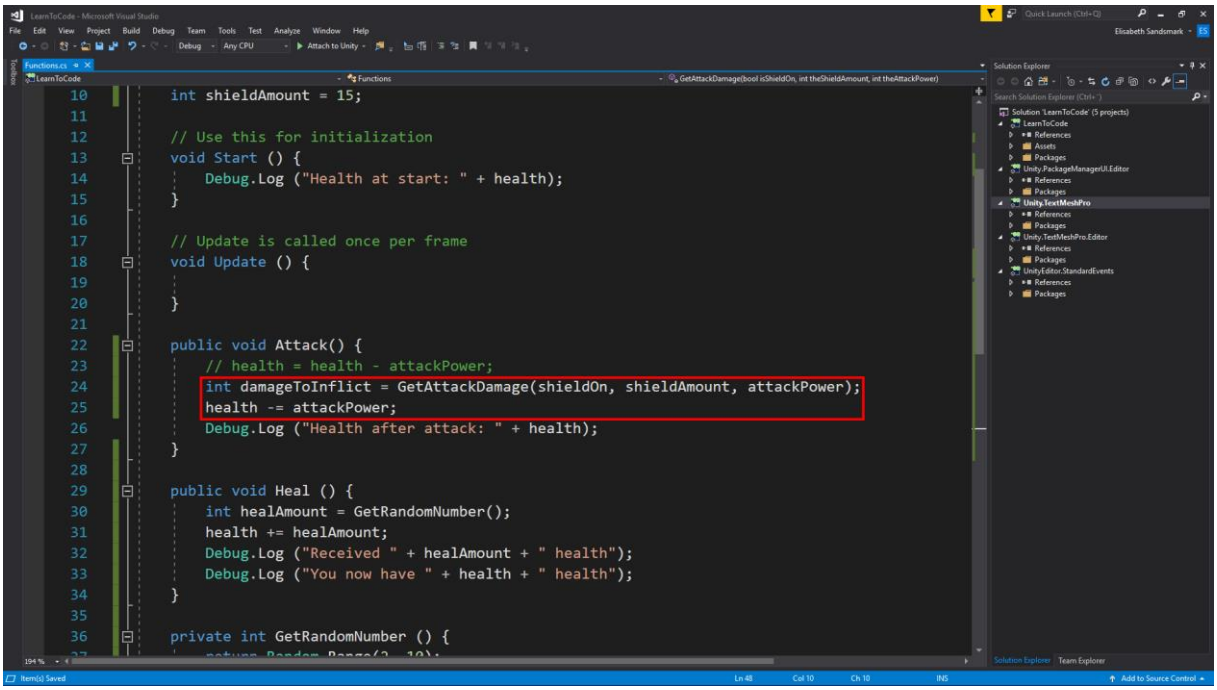
(PS: VI lager heller flere funksjoner enn å lage lange funksjoner. På denne måten blir de enklere å bruke dem om igjen i stedet for å ha funksjoner som er spesifikk til kun én ting.)

```
private int GetAttackDamage (bool isShieldOn, int theShieldAmount, int theAttackPower) {
        // Setter start damage til 0.
        int damage = 0;
        //tester om spilleren har et shield. Dersom spilleren har et shield vil ikke attack power vaere like sterk som dersom spillren ikke har et shield. Vi gjør shield amount om til en float og deretter tilbake til en int igjen.
        if (isShieldOn) {
                damage = (theAttackPower – (theShieldAmount * 10f) as int;
        }
        // Dersom spilleren ikke har et shield vil attack damage vaere den originale attack power.
        else {
        damage = theAttackPower;
        // Returnerer hvor mye damage er.
        return damage;
}
```



Vi endrer og legger til følgende kode under Attack().

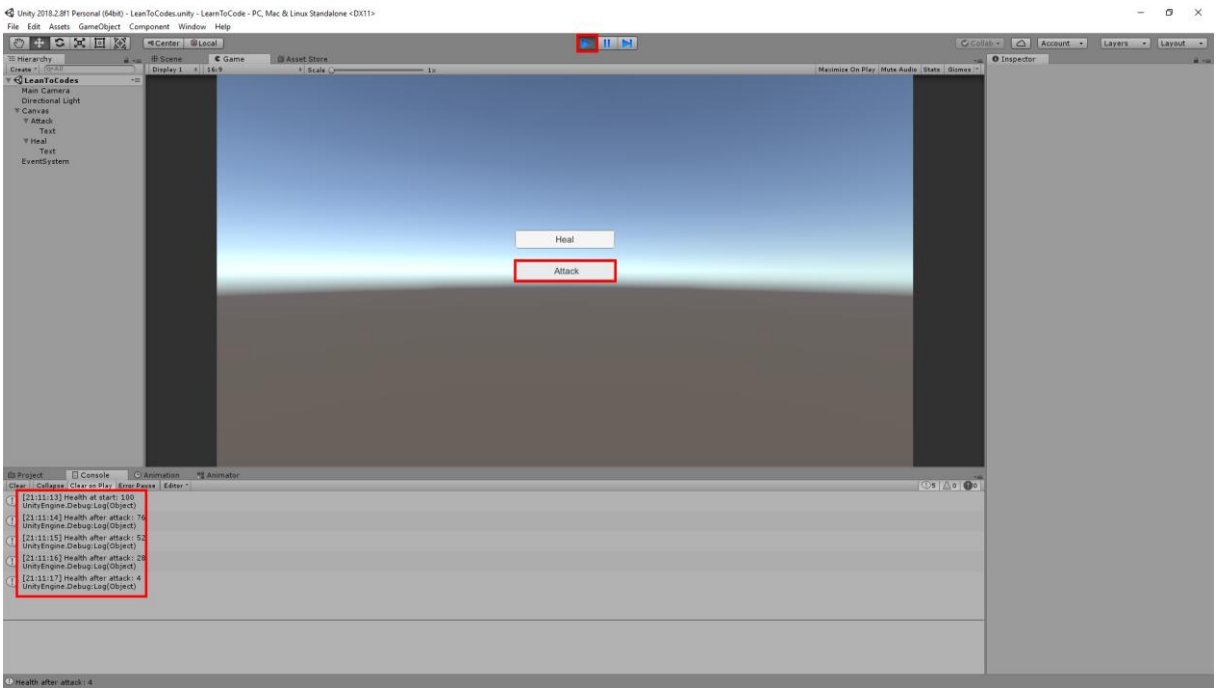
```
int damageToInflict = GetAttackDamage (shieldOn, shieldAmount, attackPower);
health -= damageToInflict;
```



```
int shieldAmount = 15;

// Use this for initialization
void Start () {
    Debug.Log ("Health at start: " + health);
}

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

}

public void Attack() {
    // health = health - attackPower;
    int damageToInflict = GetAttackDamage(shieldOn, shieldAmount, attackPower);
    health -= attackPower;
    Debug.Log ("Health after attack: " + health);
}

public void Heal () {
    int healAmount = GetRandomNumber();
    health += healAmount;
    Debug.Log ("Received " + healAmount + " health");
    Debug.Log ("You now have " + health + " health");
}

private int GetRandomNumber () {
```

Lagre og test for å sjekke om det fungerer. I stedet for at spilleren tar 25 damage, så skal han nå ta 24 damage i stedet siden han har et shield.

Matten bak: Attack power er 25. Shiled amount er 15. 15 * 10% = 1.5. Siden vi gjør en float om til iteger vil det bli 1 i stedet for 1.5. Vårt shield vil beskytte oss for 1 damage. Dette gjør at den originale damage verdien går fra 25 til 24 hver gang vi bruker attack. Ikke det mest imponerende shieldet, men det er et eksempel på hvordan vi kan bruke funksjoner.



Prøv gjerne å lage dine egne funksjoner!