Lektion 4

Grundlæggende programmering i VR

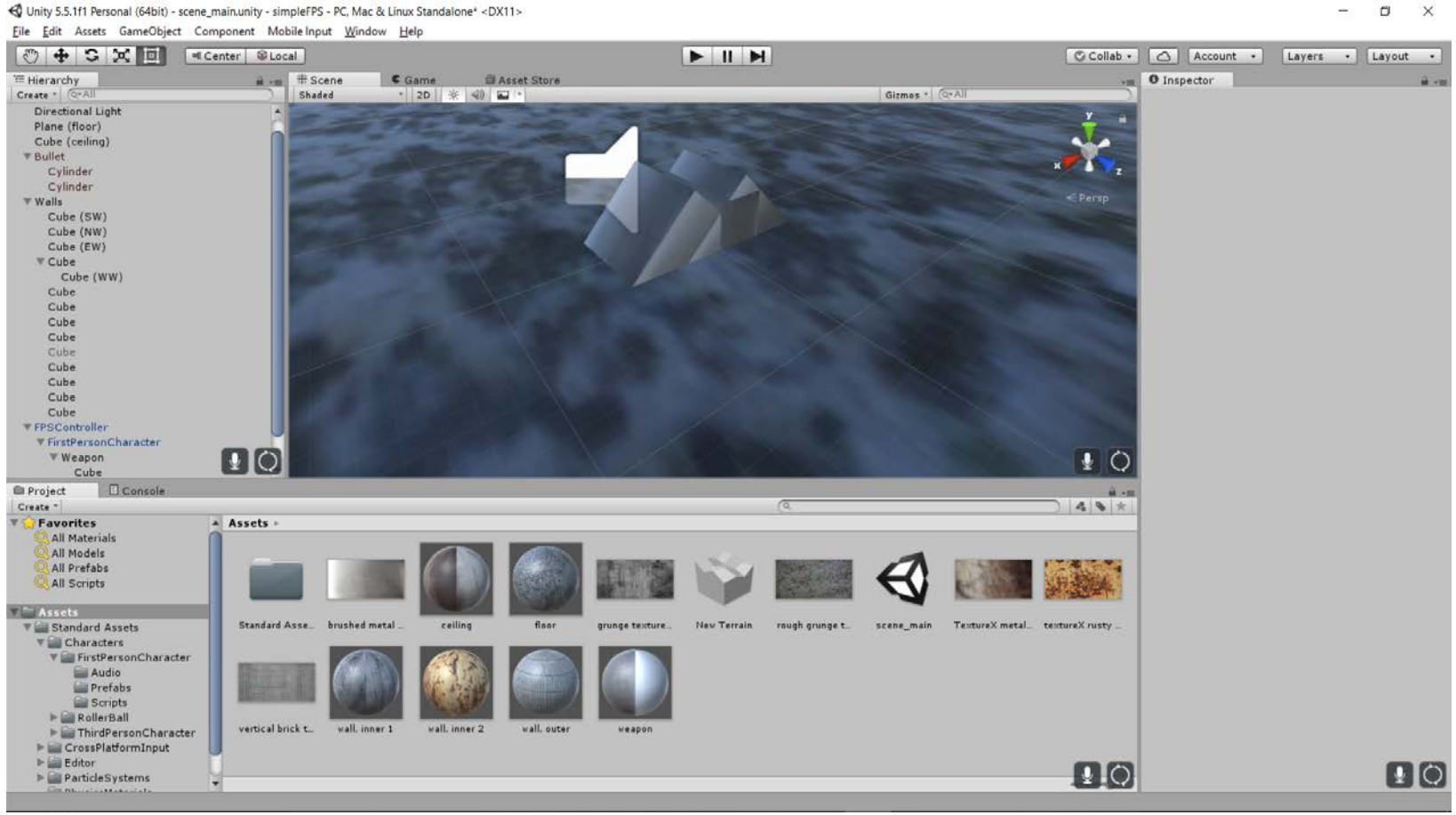
Plan for i dag

- Simpelt FPS
- C# og objekt orienteret programmering
  - Metoder
  - Loops / løkker
  - Random
- Vi koder 3D uden modeller

Simpelt FPS

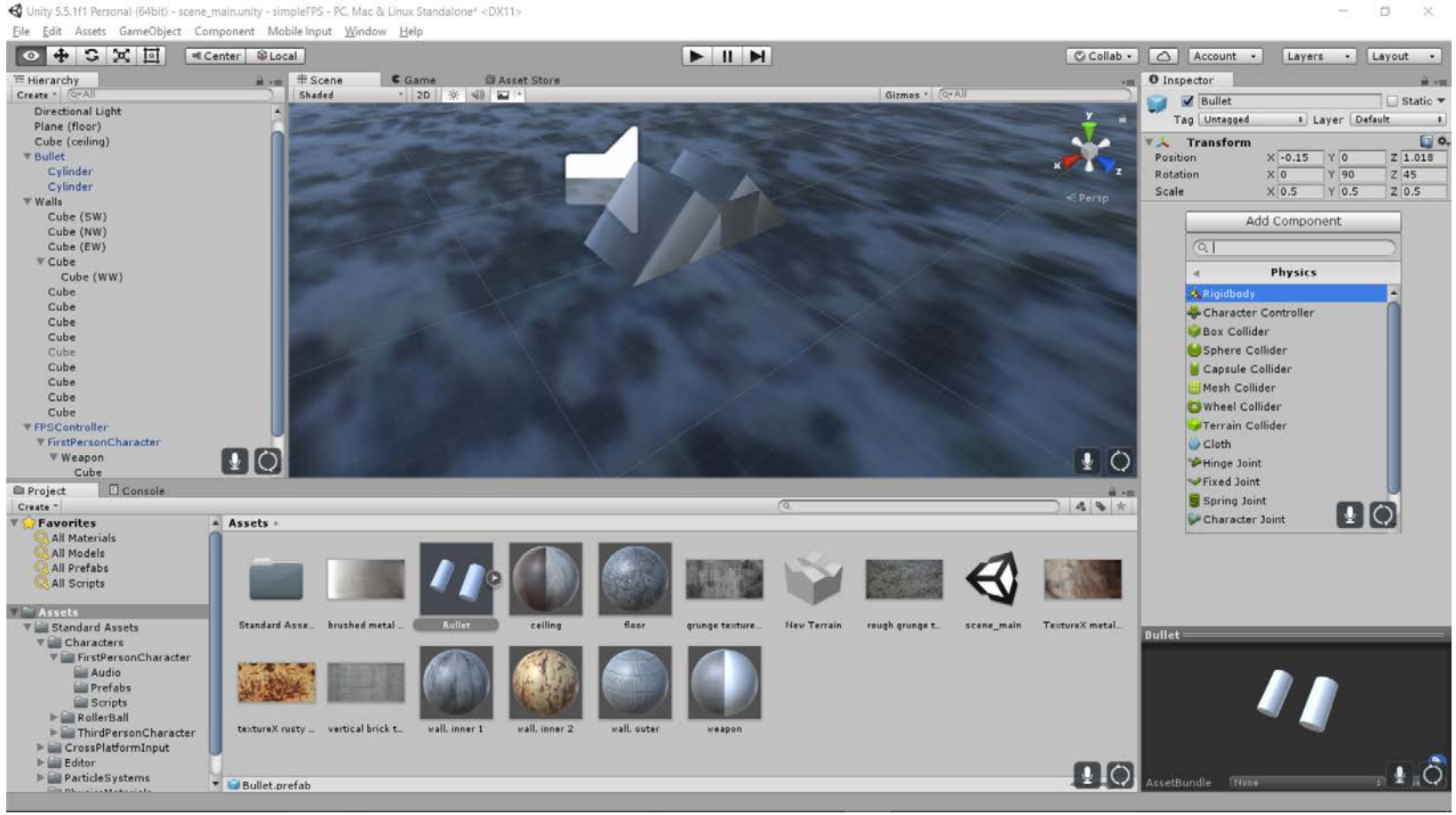
Træk kuglen fra hierarkiet ned i assets området for at gøre det til en prefab

Slet den derefter fra hierarkiet, da vi ikke skal bruge den fast



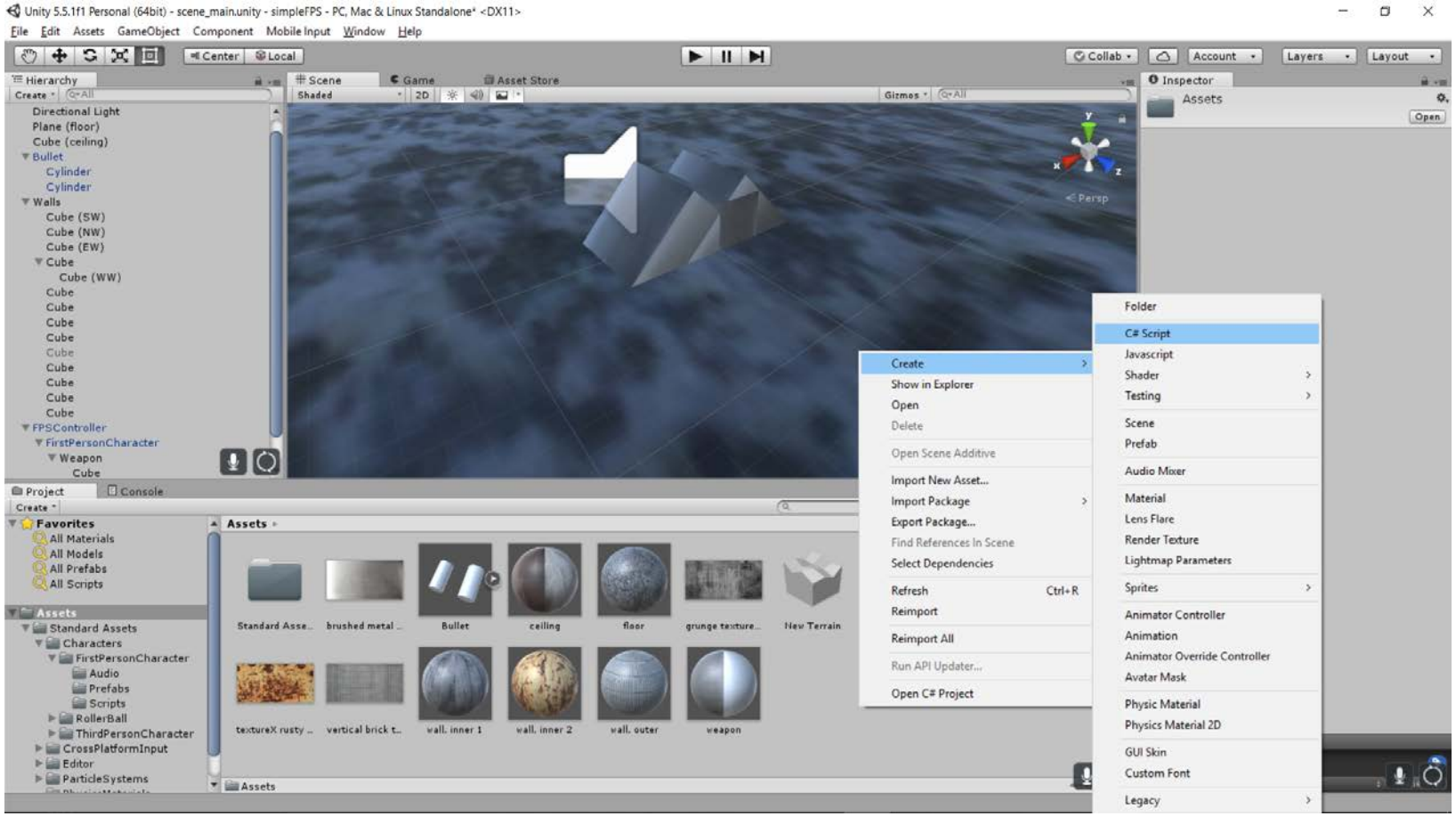
Kuglerne skal have fysiske egenskaber, så vi giver dem en rigidbody

Add component > physics > rigidbody

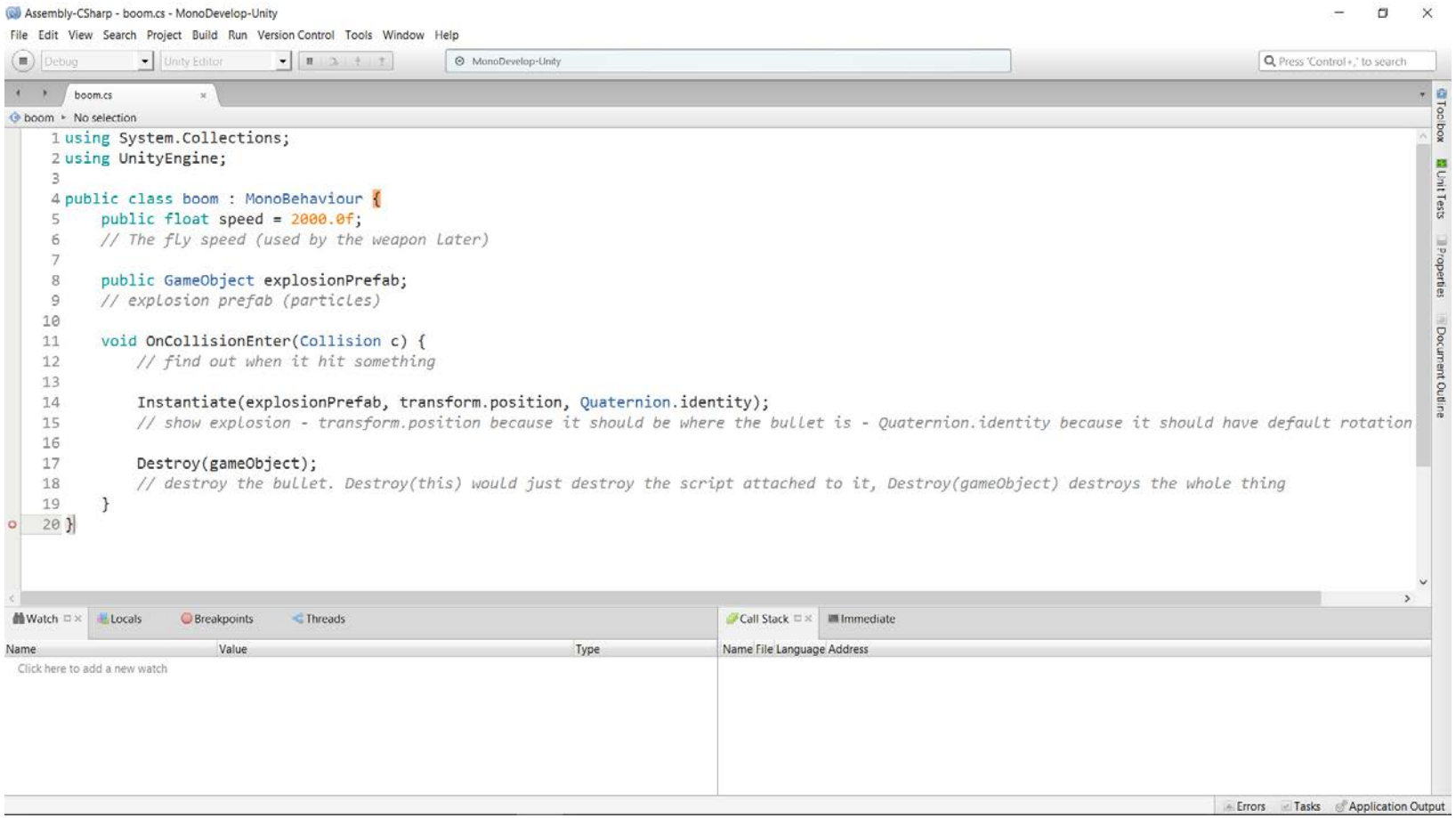


Vi skal have et script der gør at kuglerne eksploderer når de rammer

Create > C# Script



Føj koden herunder til script filen



D:\Unity\simpleFPS\Assets\boom.cs - MonoDevelop-Unity

File Edit View Search Project Build Run Version Control Tools Window Help

Debug | Unity Editor | MonoDevelop-Unity | Press 'Control+,' to search

boom.cs

boom ▸ No selection

```
1  using System.Collections;
2  using UnityEngine;
3
4  public class boom : MonoBehaviour {
5      public float speed = 2000.0f;
6      // The fly speed (used by the weapon later)
7
8      public GameObject explosionPrefab;
9      // explosion prefab (particles)
10
11     void OnCollisionEnter(Collision c) {
12         // find out when it hit something
13
14         Instantiate(explosionPrefab, transform.position, Quaternion.identity);
15         // show explosion - transform.position because it should be where the bullet is - Quaternion.identity because it should have default rotation.
16
17         Destroy(gameObject);
18         // destroy the bullet. Destroy(this) would just destroy the script attached to it, Destroy(gameObject) destroys the whole thing
19     }
20 }
```

Toolbox | Unit Tests | Properties | Document Outline

Watch | Locals | Breakpoints | Threads

Name	Value	Type

Click here to add a new watch

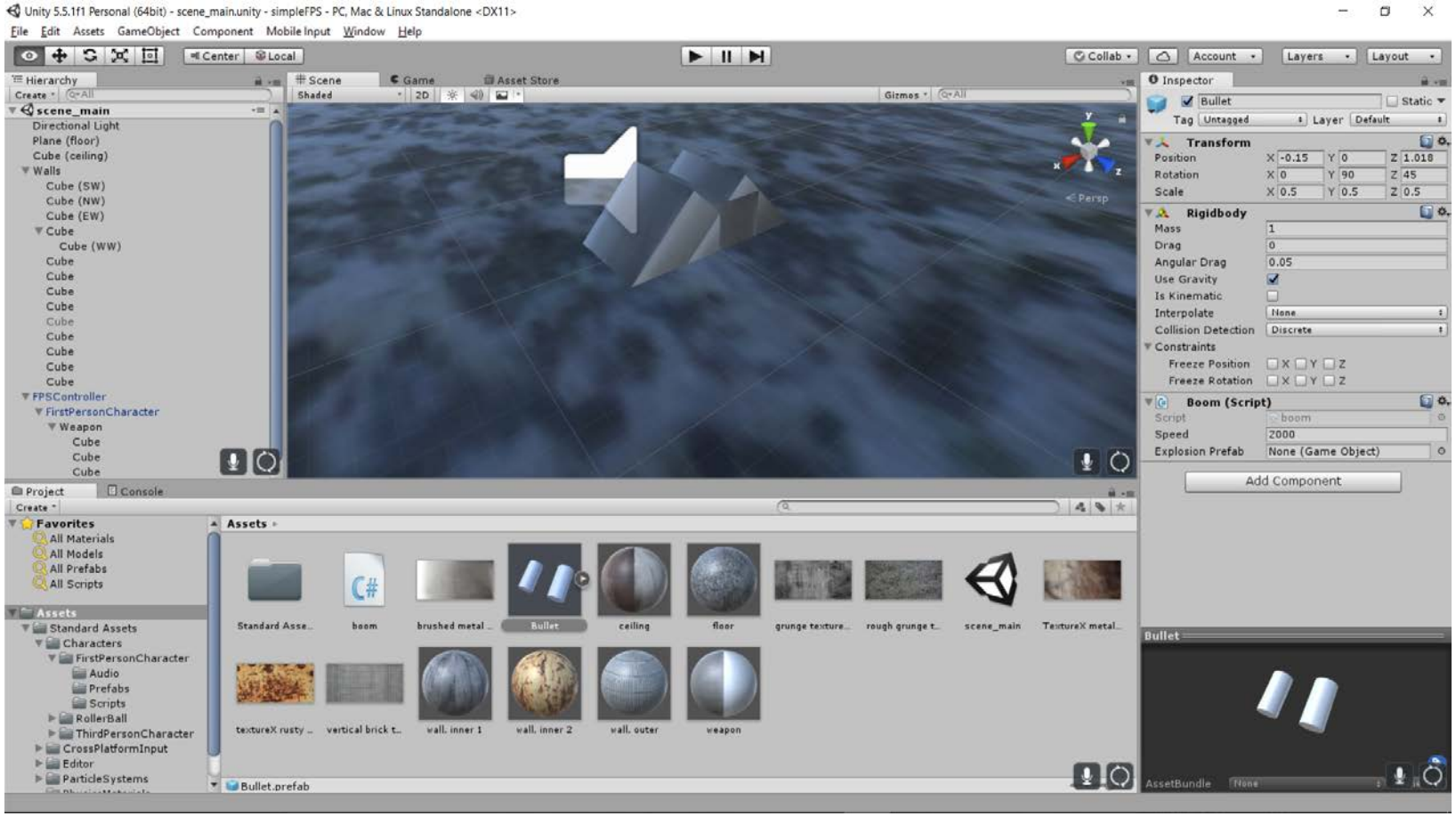
Call Stack | Immediate

Name	File	Language	Address

Errors | Tasks | Application Output

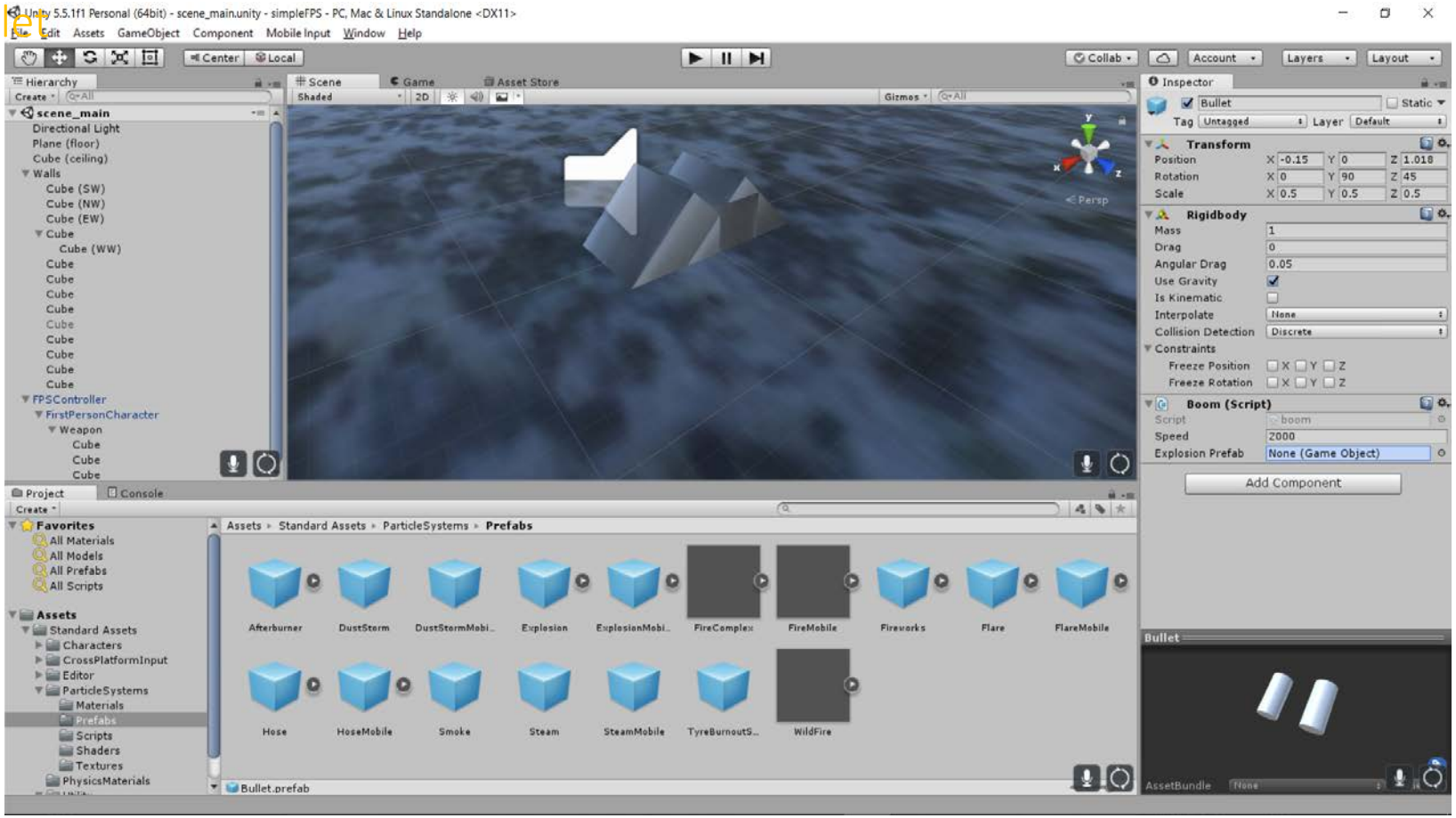
Vi instantierer eksplosionen og ødelægger det ved at tilføje scriptet

Bare træk det fra assets til indstillings-Panelet for Bullet prefab



Eksplosionen får vi gennem unitys partikler, som vi importerede i starten

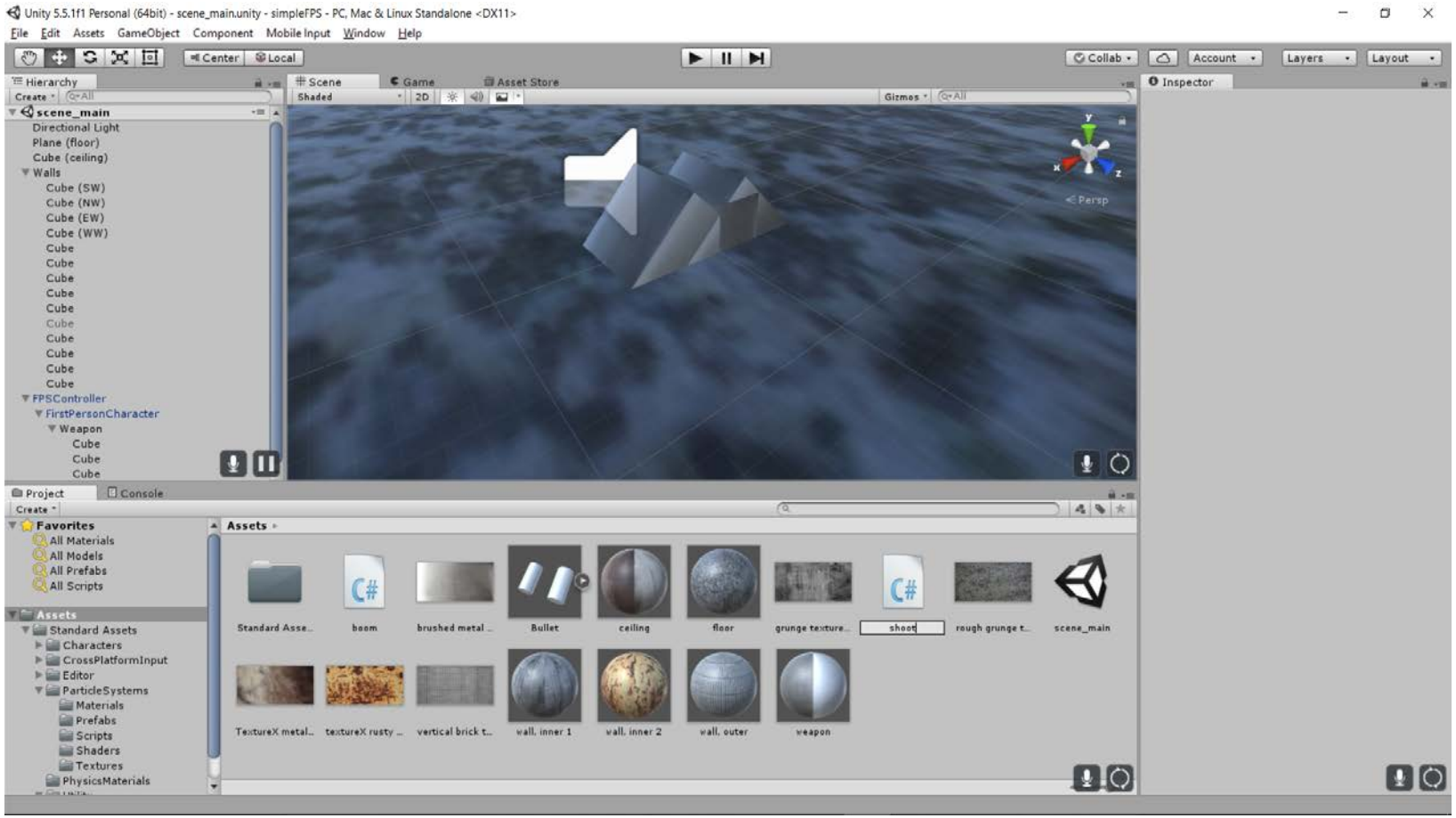
Standard Assets > Particlesystems > prefabs og træk så explosion ind i booms scriptets explosion felt på vores bullet



Sæt eventuelt scale ned til 0.2 da man ellers stort set kun kan skyde opad. Prøv dig frem med det senere når vi har spillet oppe at køre.

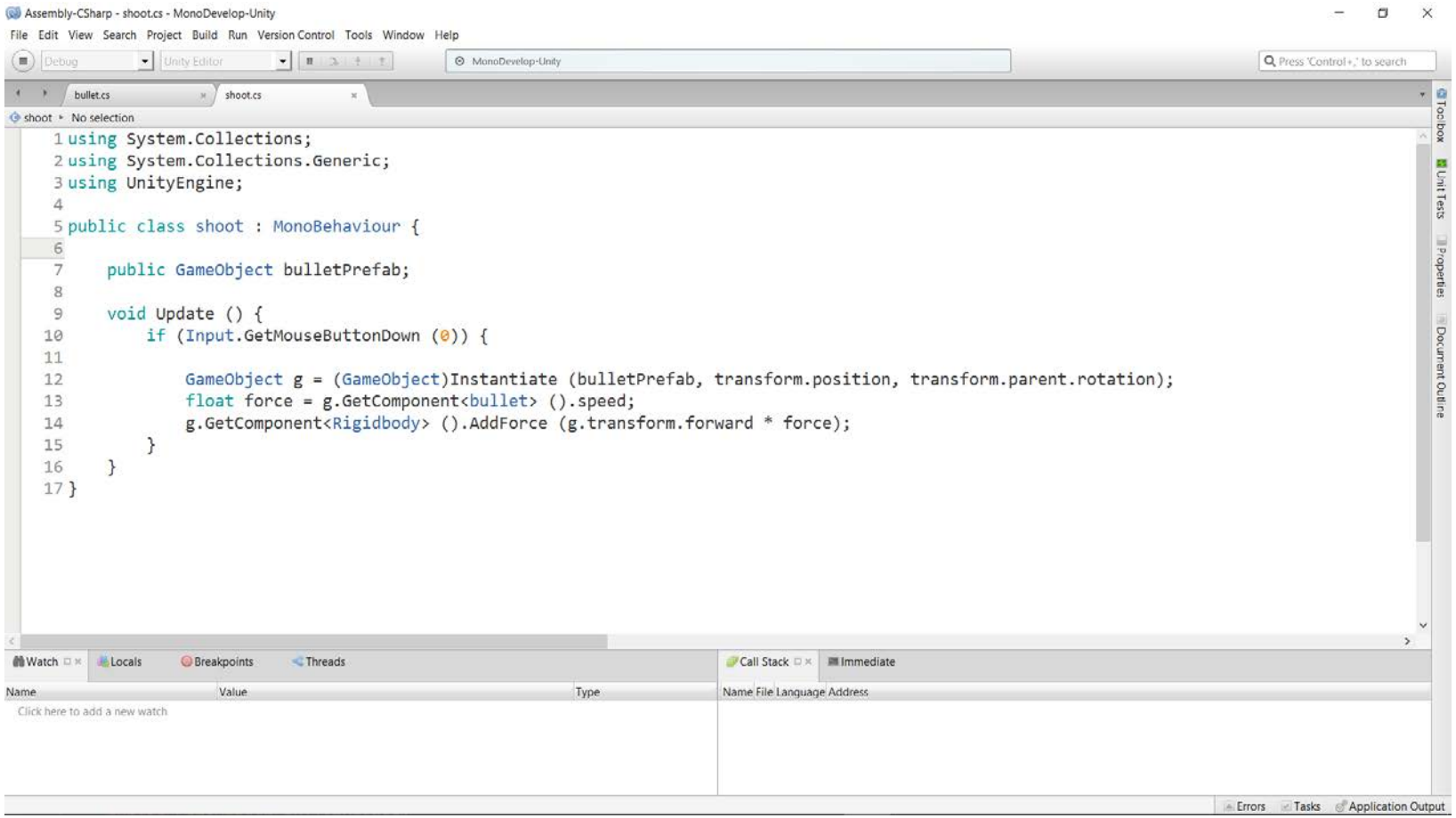
For at kuglerne kan flyve skal vi have et script til

Create > C# Script



Føj koden herunder til script filen

Vi bruger rigidbodys Addforce metode til at lade det flyve fremad



Assembly-CSharp - shoot.cs - MonoDevelop-Unity

File Edit View Search Project Build Run Version Control Tools Window Help

Debug | Unity Editor | MonoDevelop-Unity | Press 'Control+,' to search

bullet.cs | shoot.cs

shoot ▸ No selection

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class shoot : MonoBehaviour {
6
7      public GameObject bulletPrefab;
8
9      void Update () {
10         if (Input.GetMouseButtonDown (0)) {
11
12             GameObject g = (GameObject)Instantiate (bulletPrefab, transform.position, transform.parent.rotation);
13             float force = g.GetComponent<bullet> ().speed;
14             g.GetComponent<Rigidbody> ().AddForce (g.transform.forward * force);
15         }
16     }
17 }
```

Watch | Locals | Breakpoints | Threads

Name | Value | Type

Click here to add a new watch

Call Stack | Immediate

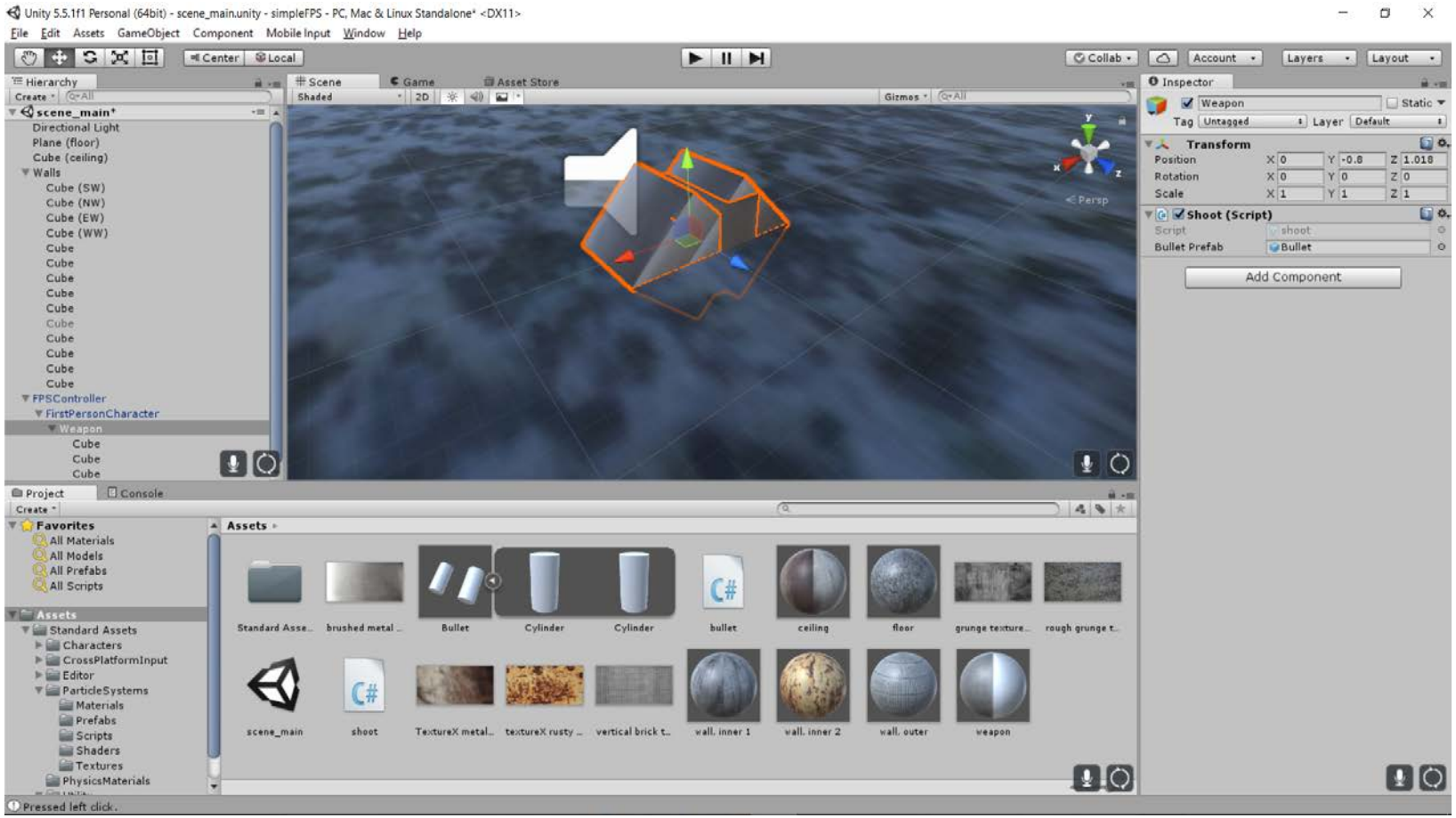
Name | File | Language | Address

Toolbox | Unit Tests | Properties | Document Outline

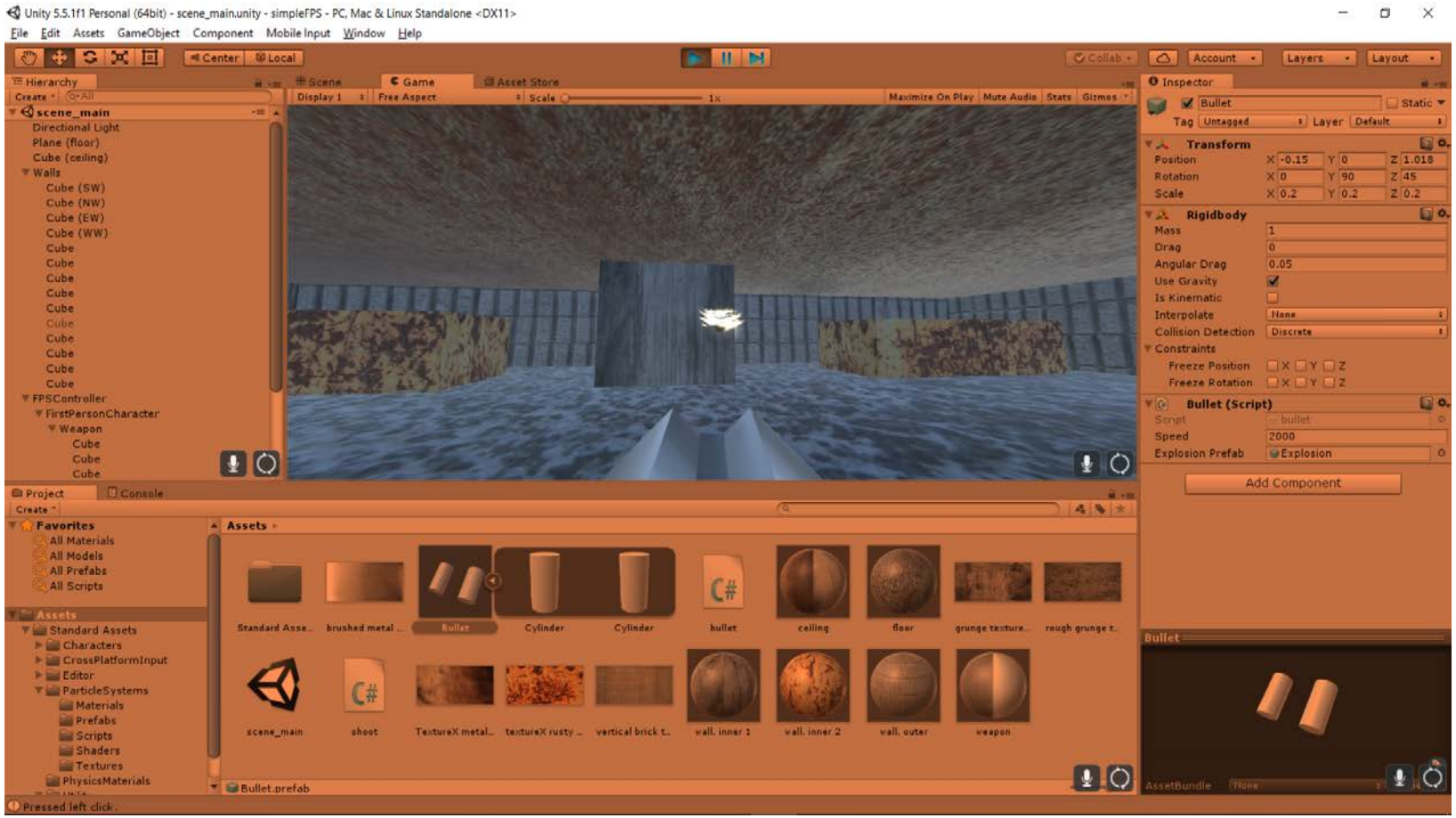
Errors | Tasks | Application Output

Føj shoot scriptet til vores våben

husk at trække vores Bullet prefab over i feltet

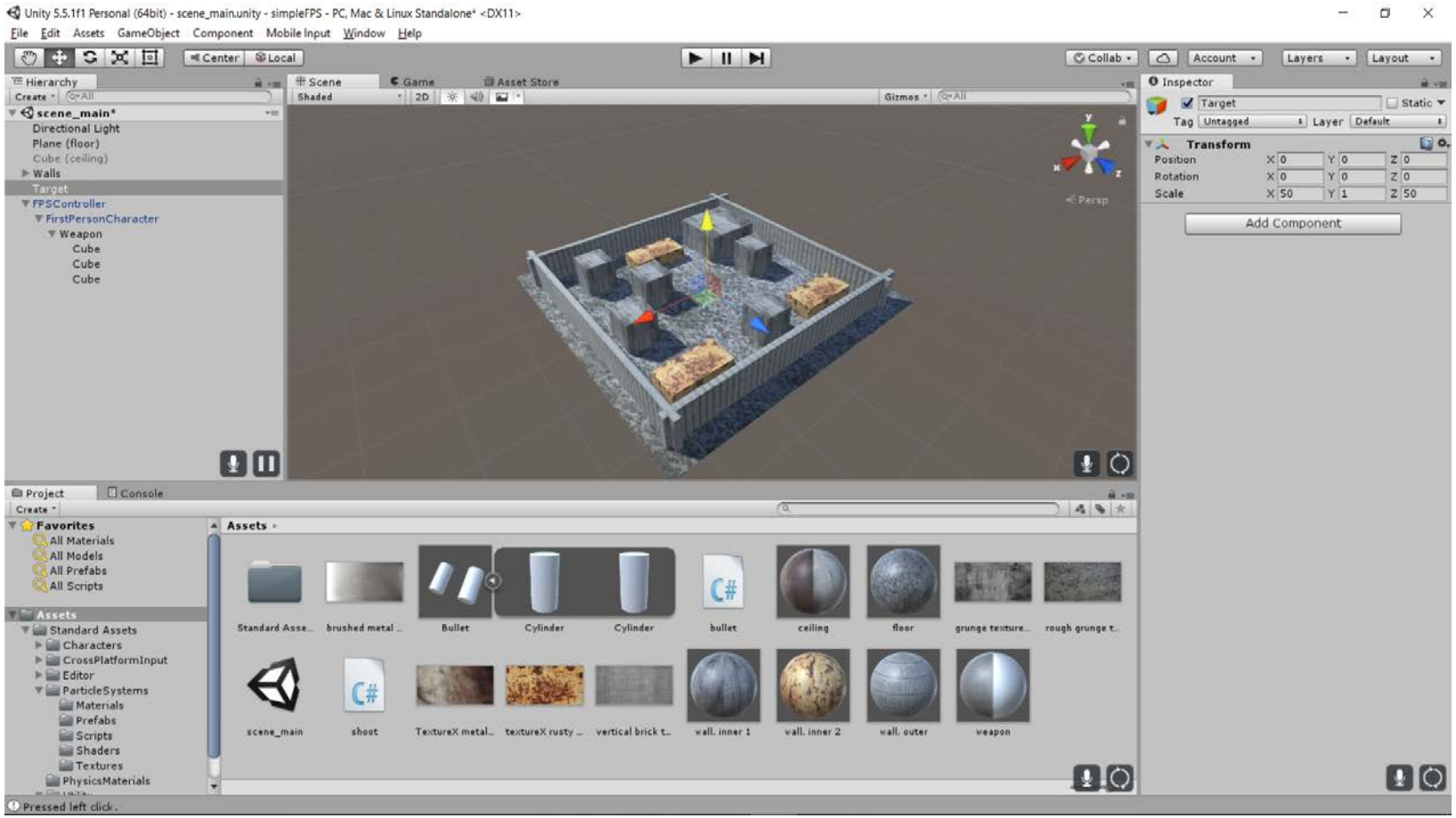


Sådan!


Unity 5.5.1f1 Personal (64bit) - scene_main.unity - simpleFPS - PC, Mac & Linux Standalone <DX11>
File Edit Assets GameObject Component Mobile Input Window Help
Hierarchy
scene_main
Directional Light
Plane (floor)
Cube (ceiling)
Walls
Cube (SW)
Cube (NW)
Cube (EW)
Cube (WW)
FPSController
FirstPersonCharacter
Weapon
Scene
Game
Asset Store
Inspector
Bullet
Transform
Position
Rotation
Scale
Rigidbody
Mass
Drag
Angular Drag
Use Gravity
Is Kinematic
Interpolate
Collision Detection
Constraints
Bullet (Script)
Speed
Explosion Prefab
Add Component
Project
Console
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Standard Assets
Characters
CrossPlatformInput
Editor
ParticleSystems
Materials
Prefabs
Scripts
Shaders
Textures
PhysicsMaterials
Bullet.prefab
Pressed left click.

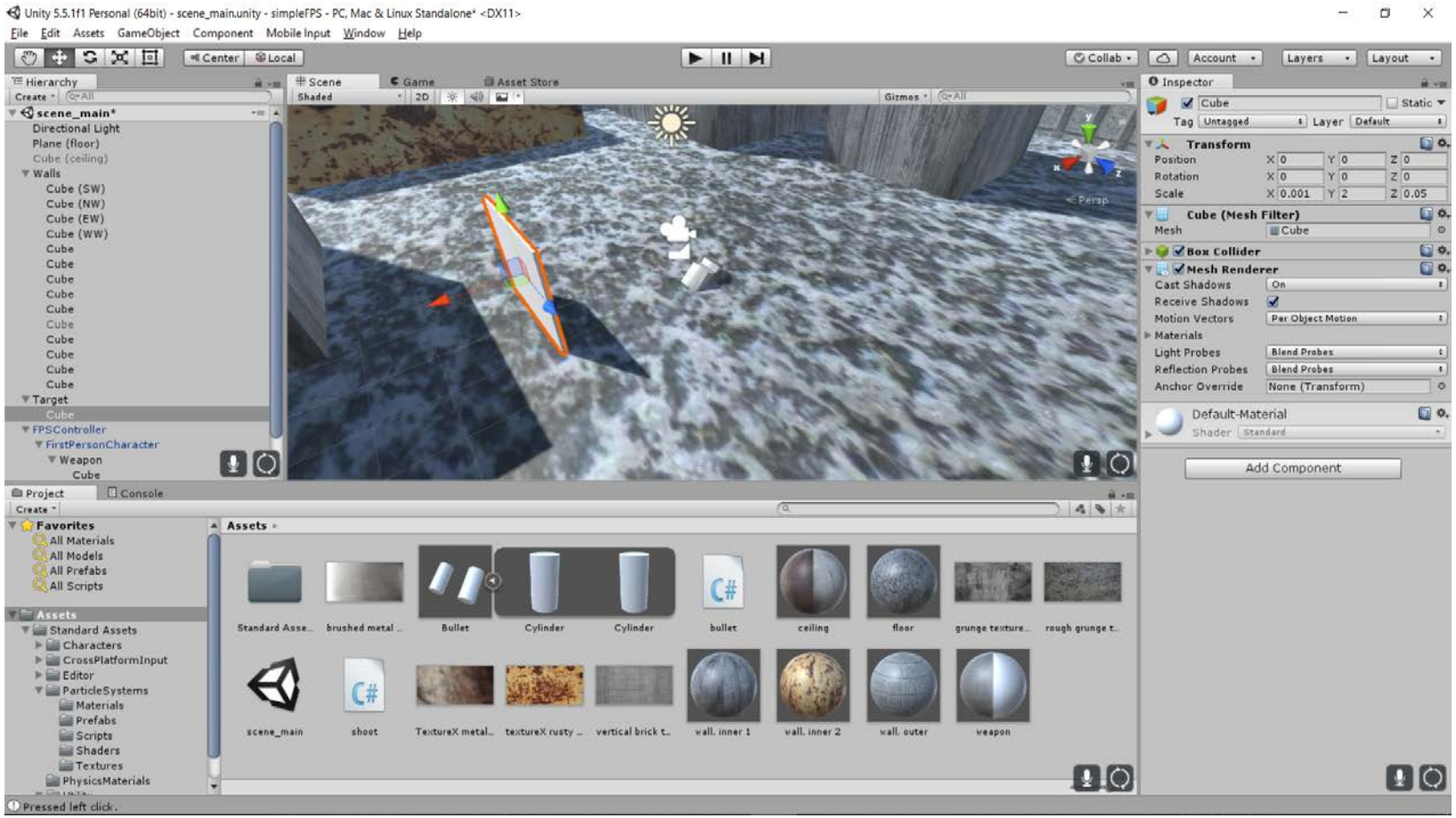
Las os føje nogle skydeskiver til

Create empty taget som overordnet



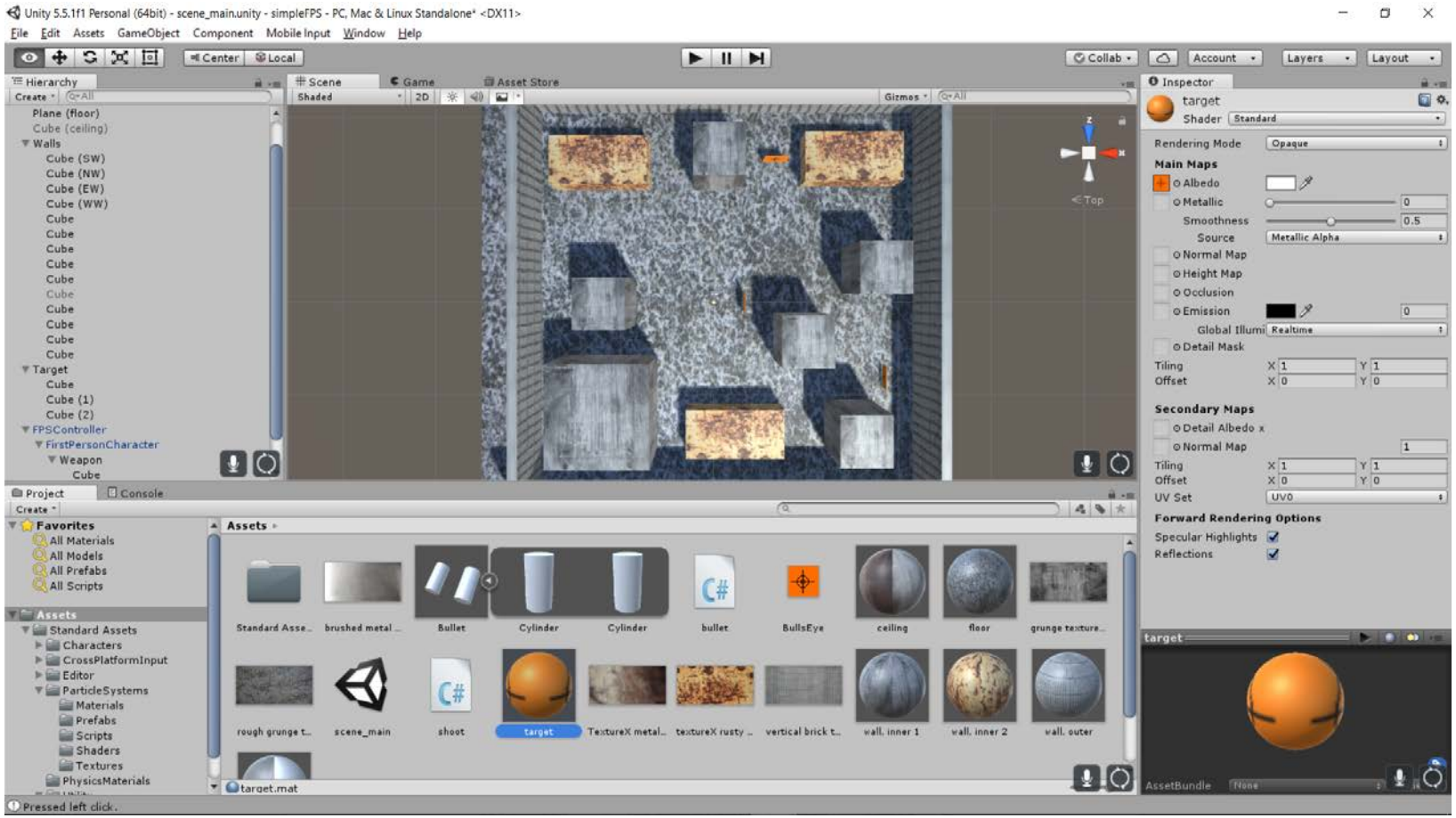
Las os føje nogle skydeskiver til

lav cubes derefter og placer dem rundt om på banen



Las os føje nogle skydeskiver til

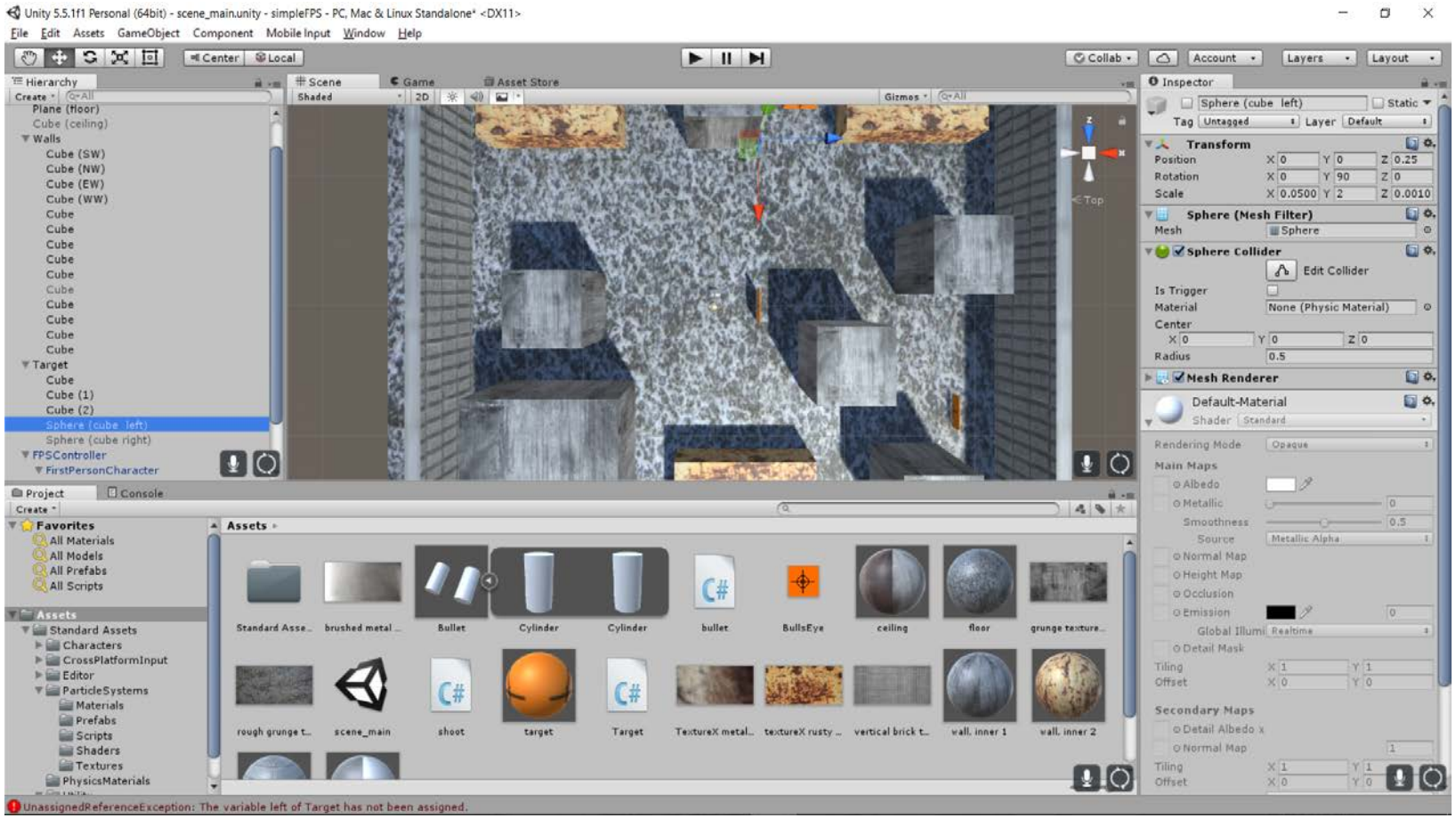
lav et passende materiale og føj det til vores skydeskiver



Lad os føje nogle skydeskiver til

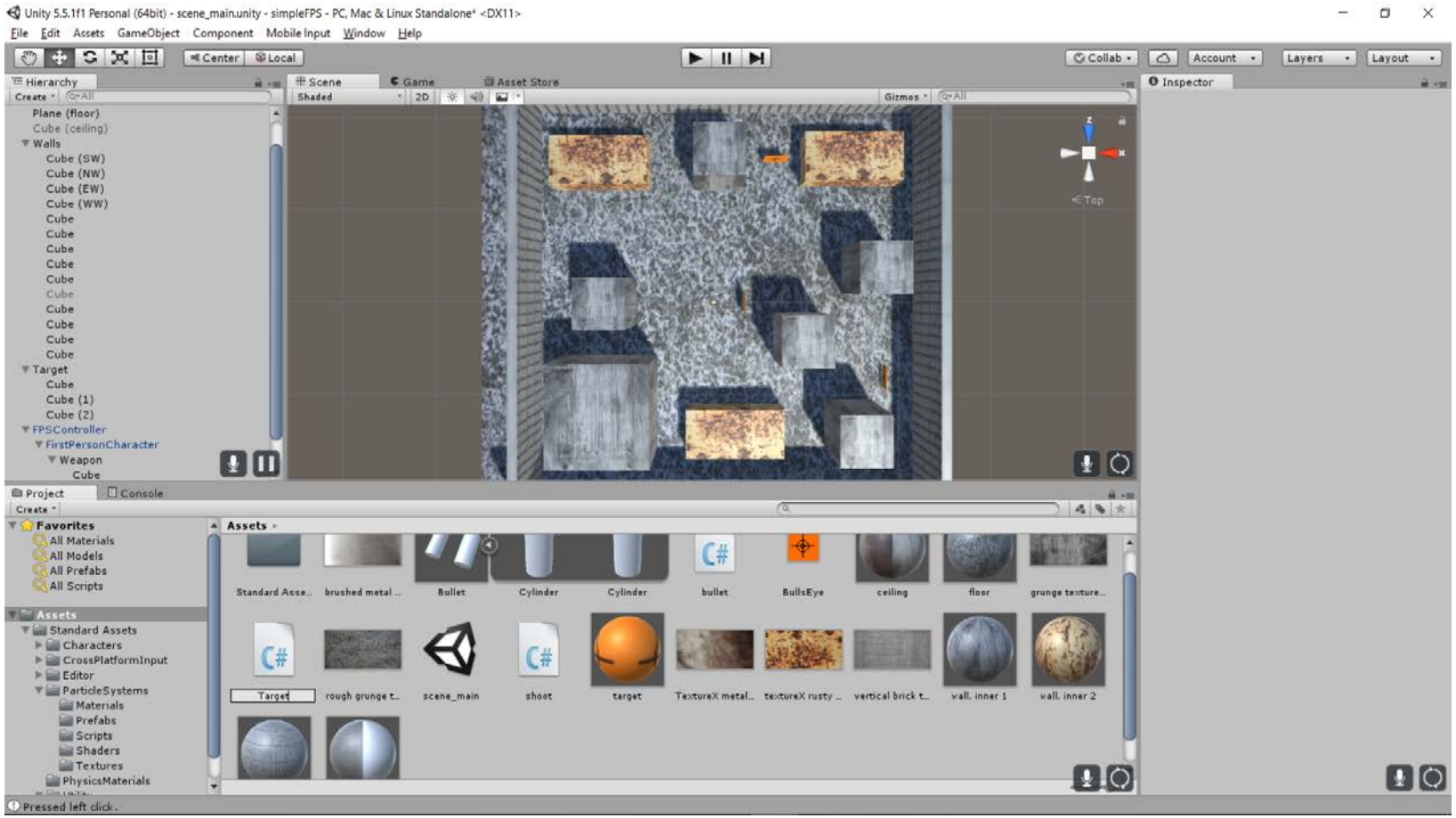
Vi skal have nogle punkter de kan bevæge sig mellem, til højre og venstre for. De behøves ikke være synlige for spilleren.

ikke



Las os føje nogle skydeskiver til

Statiske skydeskiver er kedelige, så vi laver et script der får dem til at bevæge sig



Assembly-CSharp - Target.cs* - MonoDevelop-Unity

File Edit View Search Project Build Run Version Control Tools Window Help

Debug | Unity Editor | MonoDevelop-Unity | Press 'Control+,' to search

bullet.cs | shoot.cs | Target.cs

target ▸ Update ()

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class target : MonoBehaviour {
5      // left and right marks
6      public Transform left;
7      public Transform right;
8
9      public float speed = 1.0f;  // speed
10
11     bool dir = false;   // current direction (false means to the left, true means to the right)
12
13     void Update () {
14         if (dir) {
15             // go closer to the right one
16             transform.position = Vector3.MoveTowards(transform.position, right.position, Time.deltaTime * speed);
17
18             // reached it?
19             if (transform.position == right.position)
20                 dir = !dir; // go to opposite direction next time
21         } else {
22             // go closer to the left one
23             transform.position = Vector3.MoveTowards(transform.position, left.position, Time.deltaTime * speed);
24
25             // reached it?
26             if (transform.position == left.position)
27                 dir = !dir; // go to opposite direction next time
28         }
29     }
30 }
```

Watch | Locals | Breakpoints | Threads

Name	Value	Type

Click here to add a new watch

Call Stack | Immediate

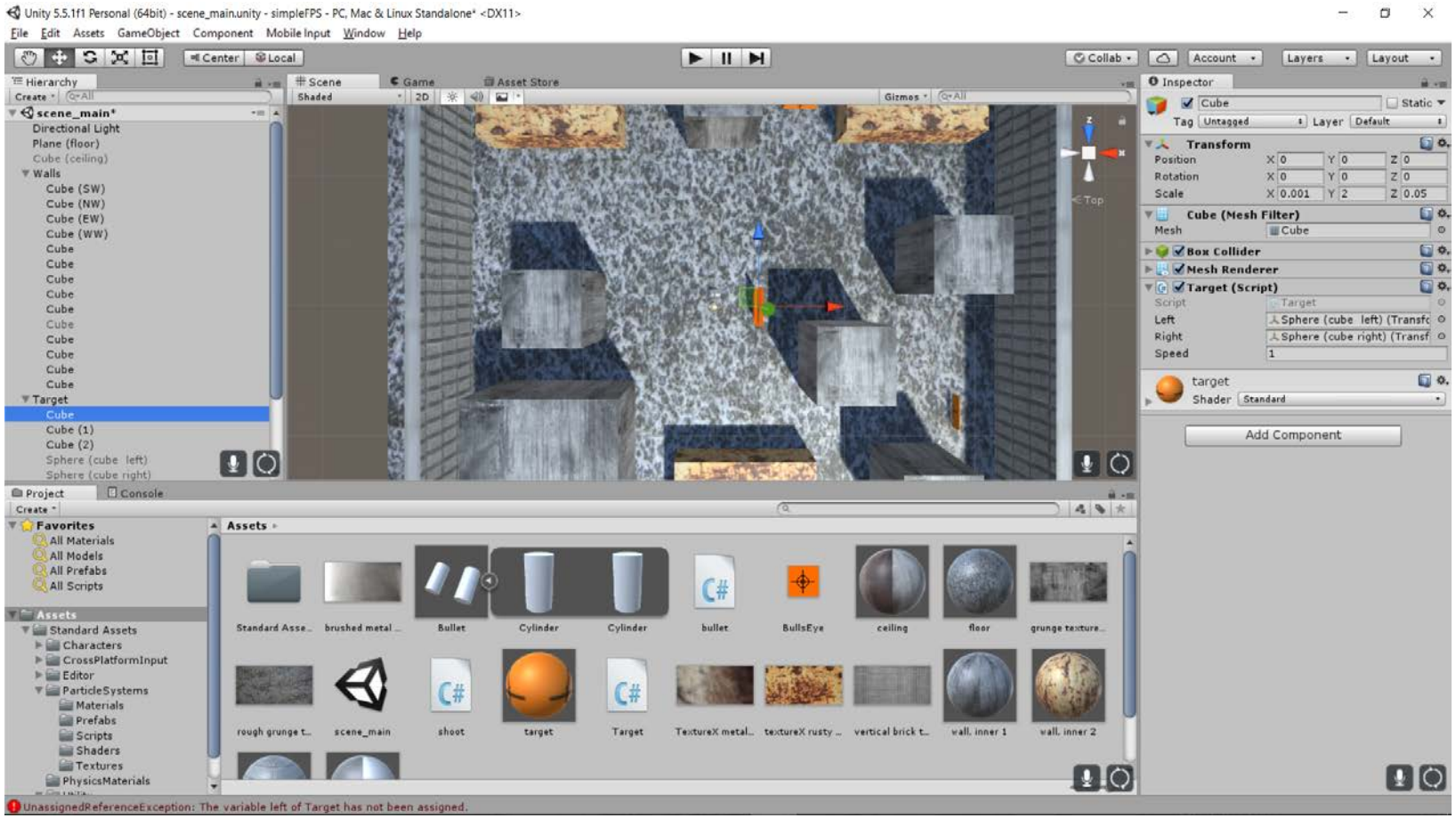
Name	File	Language	Address

Toolbox | Unit Tests | Properties | Document Outline

Errors | Tasks | Application Output

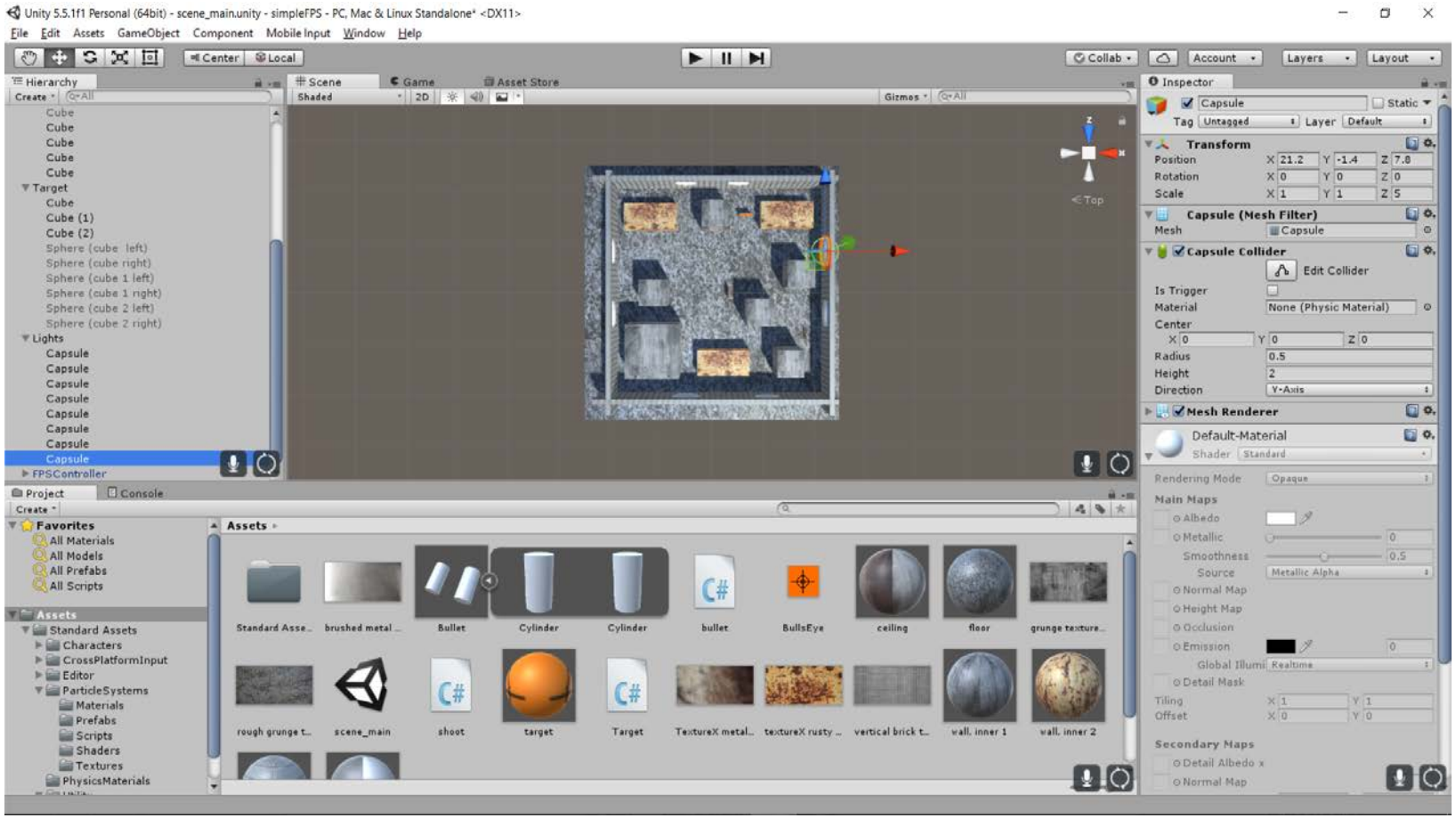
Las os føje nogle skydeskiver til

Føj scriptet til skydeskiverne og sæt højre og venstre mærkerne



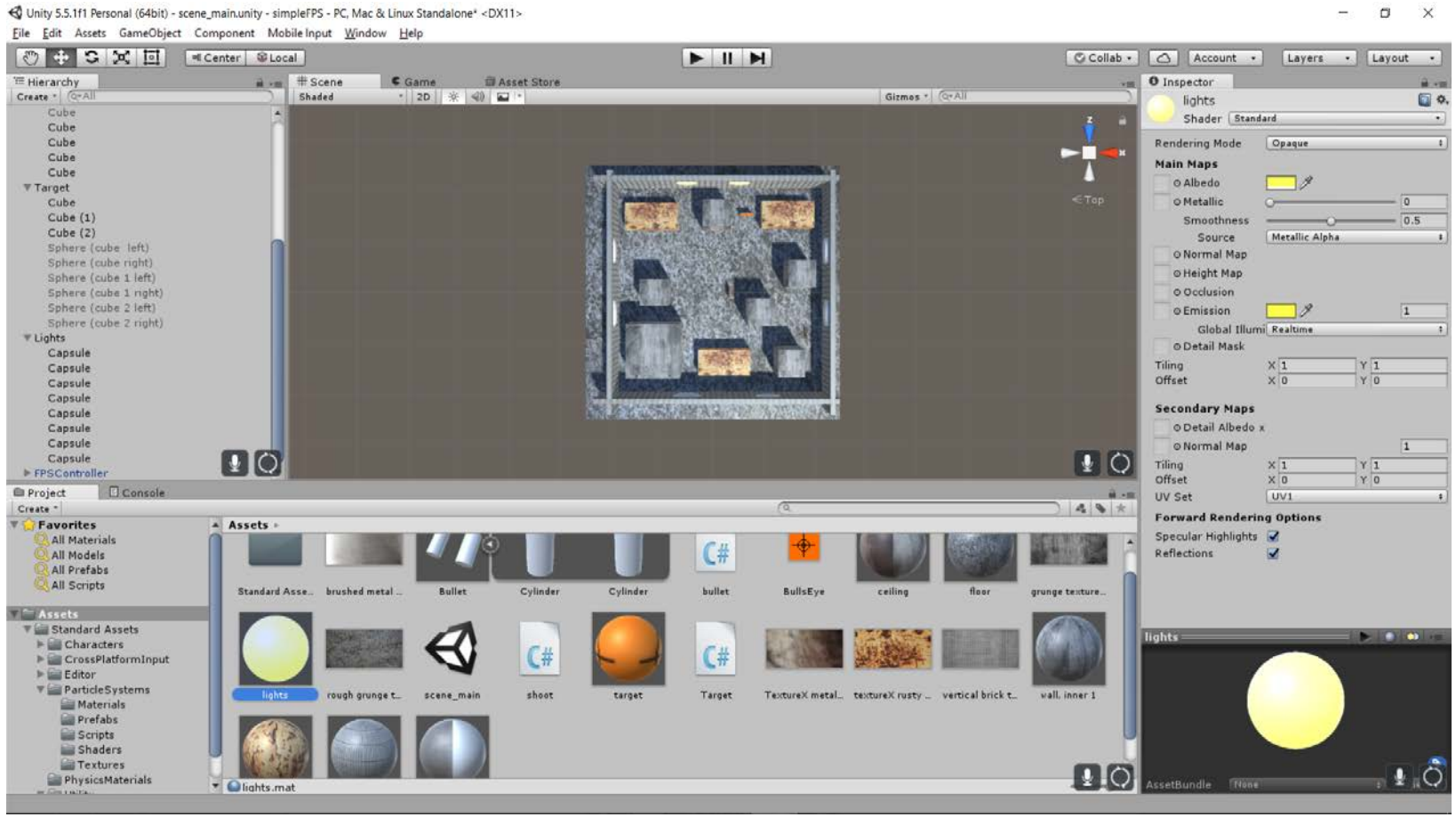
Med loftet aktivt er scenen lidt mørk

Vi laver en gruppe, lights, der indeholder capsules til lysstofrør



Med loftet aktivt er scenen lidt mørk

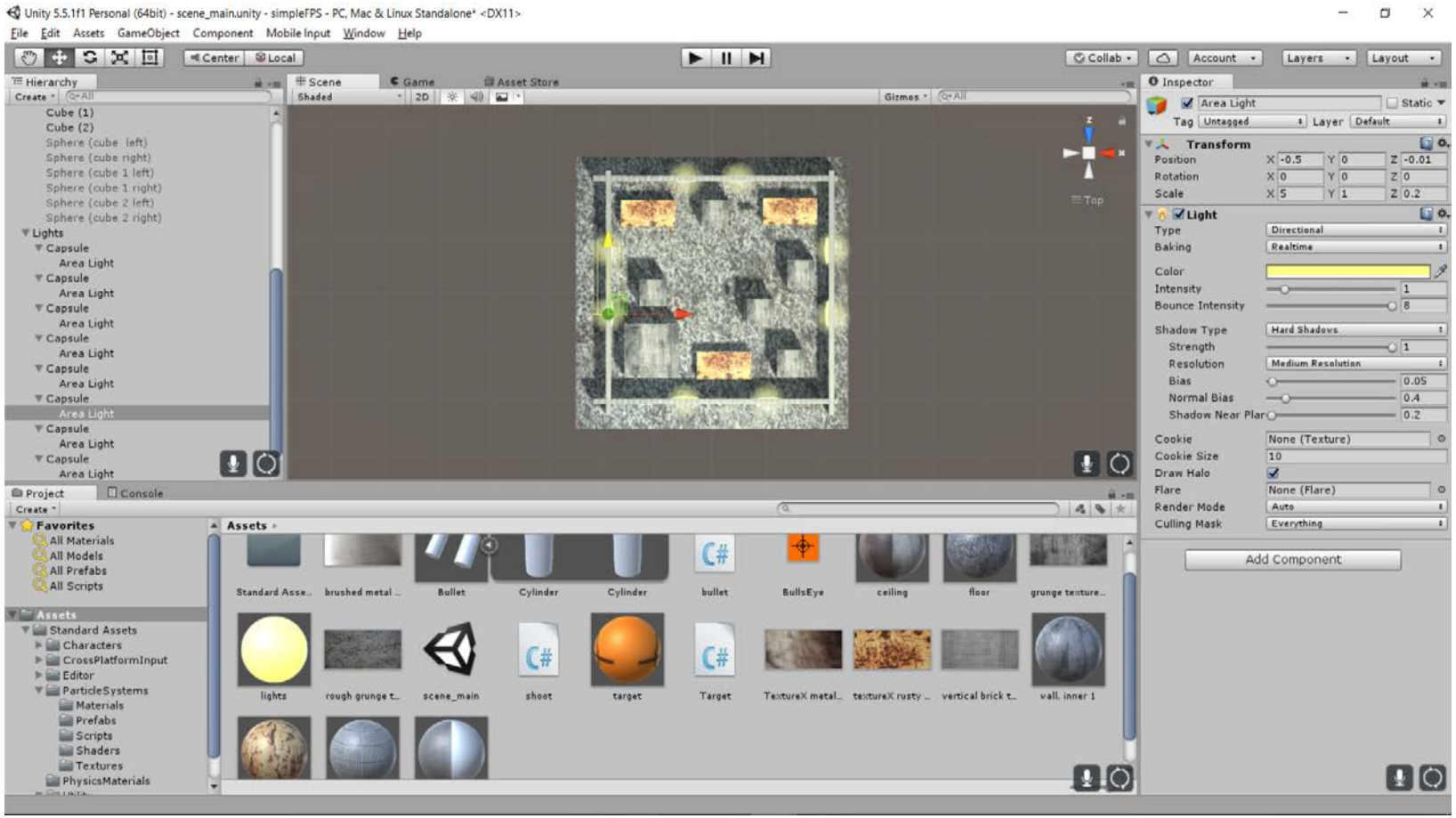
Vi laver et nyt materiale så vores lysstofrør er mere gule



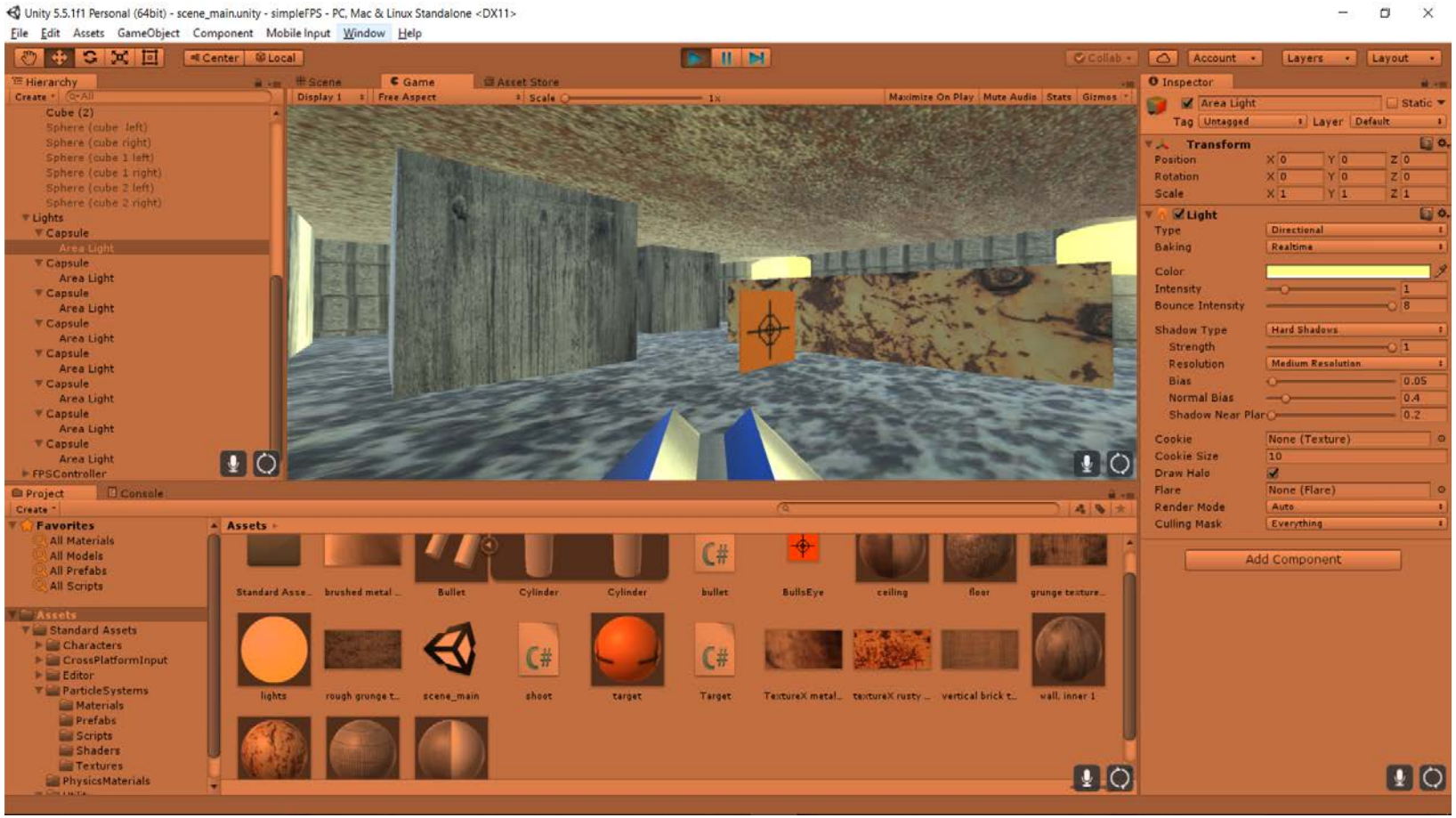
Med loftet aktivt er scenen lidt mørk

Vi føjer area lights til og lægger dem oveni vores ”lysstofrør”

Eksperimenter med indstillingerne for det bedste resultat



Måske lidt meget men sådan!



C# og objekt orienteret programmering

Metoder, loops / løkker, random

Metoder

Sekvenser af kode om samme emne

Vi har allerede brugt dem men vi vender det lige igen

Metoder

- En metode er en måde at samle en sekvens af statements der udfører en bestemt handling eller beregner et bestemt resultat.
  - Dette giver større struktur og organisation for de statements, der opbygger et program.
- Hvert C # program har mindst en klasse med en metode kaldet Main.
- For at benytte en metode skal man:
  - Definere metoden.
  - Kalde metoden.
- Vi har brugt metoder hidtil, primært main() metoden, hvori hele vores programmer lå, men med mere komplekse programmer kan det være praktisk eller nødvendigt at benytte flere metoder.

Metoder

- En metodes grund struktur er

```
<Access Specifier> <Return Type> <Method Name>(Parameter List)
{
  Method Body
}
```

- Access Specifier (scope): Bestemmer synligheden af en variabel eller en metode for andre klasser.
- Return type: En metode kan returnere en værdi. Hvis metoden ikke returnerer nogen værdier, så er retur typen ugyldig (void).
- Method name: Metode navn er et entydigt id, og det er følsomt overfor store og små bogstaver. Det kan ikke være det samme som en anden identifikator erklæret i klassen.
- Parameter list (parametre og argumenter): Lukket inde mellem parenteser finder man de parametre, der anvendes til at sende og modtage data fra metoden. Parameterlisten refererer til typen, rækkefølgen og antallet af parametre i metoden. Parametre er valgfri, det vil sige at en metode ikke behøves have nogen parametre.
- Method body: Metodens krop indeholder det sæt af instruktioner, der er nødvendige for at gennemføre den ønskede aktivitet.

Metoder

```
1   using System;
2
3   namespace _1___metoder
4   {
        2 references
5       public class NumberManipulator
6       {
            1 reference
7           public int FindMax(int num1, int num2)
8           {
9               int result;
10              if (num1 > num2)
11                  result = num1;
12              else
13                  result = num2;
14
15              return result;
16          }
17      }
            0 references
18          class Program
19      {
            0 references
20          static void Main(string[] args)
21          {
22              int a = 100;
23              int b = 200;
24              NumberManipulator n = new NumberManipulator();
25              int ret;
26              ret = n.FindMax(a, b);
27              Console.WriteLine("Max value is : {0}", ret);
28              Console.ReadKey();
29          }
30      }
31  }
```

Metoder

- En metode er *ikke* et statement selvom man kunne tro det.
  - System.Console .WriteLine ("! Hej {0}", System.Console.ReadLine ());
    er et enkelt statement, der indeholder to metodeopkald. Et statement indeholder ofte et eller flere udtryk, og i dette eksempel er to af disse udtryk metodekald. Derfor danner metodekald dele af statements.
  - Selvom kodning af flere metodekald i et enkelt statement ofte vil reducere mængden af kode, betyder det ikke nødvendigvis at det øger læsbarheden, og det giver sjældent en betydelig ydelses fordel. Udviklere bør altid favorisere læsbarheden over kortfattethed.

Metoder – Refactoring

- Det at flytte en række udsagn ind i en metode i stedet for at lade dem være inline i en større metode er en form for refactoring.
- Refactoring reducerer kode dobbeltarbejde, fordi man kan kalde metoden fra flere steder i stedet for at duplikere koden.
- Refactoring øger dermed også kodens læsbarhed.
- Som del af kodning processen det er en bedste praksis øbende at gennemgå ens kode og se efter muligheder for at refactore.
  - Dette indebærer at blokke af kode, der er vanskelige at forstå og få et overblik over, flyttes ind i en metode med et navn, der klart definerer kodens adfærd.
  - Denne praksis er ofte foretrukket frem for at kommentere en blok af kode, fordi metodens navn tjener til at beskrive, hvad implementeringen gør.

Metoder – Namespaces

- Som I har bemærket begynder vi alle C# programmer med at definere et namespace.
- Namespaces er en kategoriserings mekanisme til gruppering af alle typer relateret til et bestemt område af funktionalitet.
- Namespaces er hierarkiske og kan have vilkårligt mange niveauer i hierarkiet, omend namespaces med mere end en halv snes niveauer er sjældne.
- Typisk begynder hierarkiet med et firmanavn, og derefter et produktnavn, og derefter den funktionelle område.
  - For eksempel i Microsoft.Win32.Networking, er det yderste navneområde Microsoft, som indeholder et indre navneområde Win32, som igen indeholder endnu et mere dybt indlejret Networking namespace.
  - Det burde virke bekendt i forhold til using i starten af vores programmer.

Metoder – Namespaces

- Namespaces bruges primært til at organisere typer efter område af funktionalitet så de lettere kan findes og forstås.
- De kan dog også anvendes til at undgå Typenavn kollisioner. Compileren kan f.eks. skelne mellem to typer med navnet Button så længe hver type er under forskellige namespaces.
  - Således er System.Web.UI.WebControls.Button og System.Windows.Controls.Button forskellige.
- Det er ikke altid nødvendigt at angive et namespace når du kalder en metode. Compileren kan f.eks. udlede at hvis det kaldte udtryk optræder i det samme namespace som den kaldte metode er
namespace det samme som det, der indeholder den type.

God programmerings skik

- Benyt PascalCasing (hvert ord begynder med stort) til namespace navne.
- Overvej at organisere bibliotekets hierarki for kildekode filer så det matcher namespace hierarkiet.

Metoder – Using

- Fulde namespace navne bliver hurtigt lange og klodsede, så man kan ”importere” indholdet af et eller flere namespaces ind i en fil så man ikke behøves angive de fulde navne.

```
using System;
…
Console.WriteLine("Hello, my name is Inigo Montoya");
```

- Eksemplet lader os benytte .Console fra System uden at skulle angive det forrest.
- Du kan dog ikke benytte elementer fra child namespaces, så hvis man vil have fat i StringBuilder fra System.Text namespacet skal det også kobles på med using System.Text; direktivet eller man må angive typen som System.Text.StringBuilder – ikke bare Text.Stringbuilder!

Metoder – Main

Main metode er indgangen til en C# konsol applikation eller et Windows-program. (Biblioteker og tjenester kræver ikke en Main metode som indgang.). Når programmet startes er Main metoden den første metode, der invokes.

- Main metode er indgangen i et .exe program; det er hvor program styring starter og slutter.
- Main erklæres inde i en klasse eller struct. Main skal være static (kan ikke instantieres men kun bruges som den er), og den bør ikke være offentlig. Den omsluttende klasse eller struct er ikke forpligtet til at være statisk.
- Main kan enten have en void eller int return type.
- Main metoden kan erklæres med eller uden en string [] parameter, der indeholder kommandolinjeargumenter.
  - Når du bruger Visual Studio til at oprette Windows Forms applikationer, kan du tilføje parameteren manuelt eller bruge Environment klassen til at få fat i kommandolinjeargumenter. Parametre læses som nul-indekserede kommandolinjeargumenter.
    I modsætning til C og C ++, er navnet på programmet ikke behandlet som det første kommando-line argument.


C#
#8
BASICS

Løkker

- En løkke er når en blok kode køres flere gange.
- Termen loop body henviser til et statement, typisk en kode blok, der køres i et loop indtil afslutnings-kravet er mødt.

Forskellige former for løkker

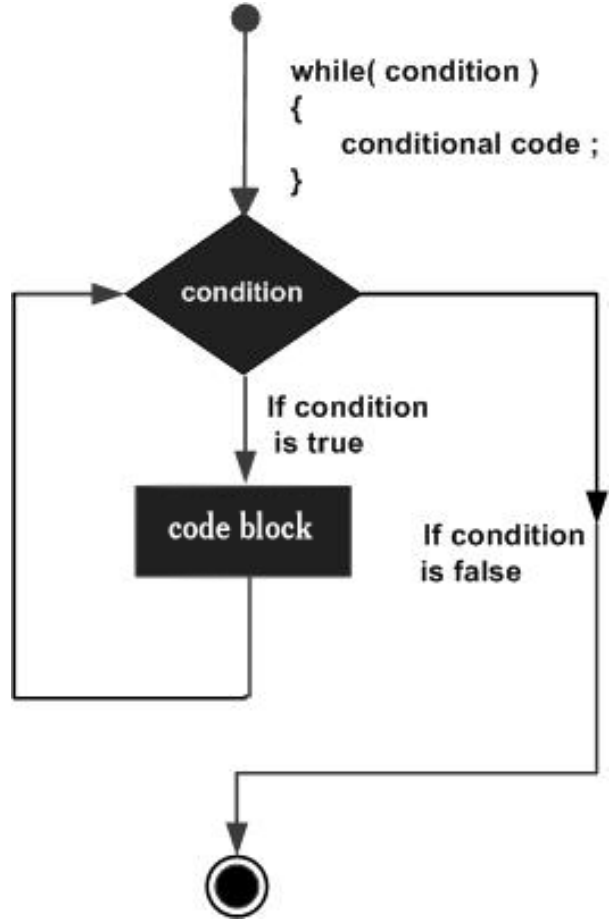
- Man bruger while til at gentage (iterate) så længe kravet er true.
- En for løkke bruges mest hensigtsmæssigt, når antallet af gentagelser er kendt, såsom når der tælles fra 0 til n.
- En do/while ligner en while-løkke, men den vil altid udføre løkken kroppen mindst én gang.

Løkker – While

While løkken er det enkleste betingede loop. Den almindelige form af while-sætningen er:

while (*condition*)
statement

Loopet kører det statement, der danner kroppen i udtrykket, rundt så længe at conditionen (der *skal* være boolean) er true.
Hvis den bliver *false* dropper udførelsen af koden kroppen og går videre til koden efter loop statementet.

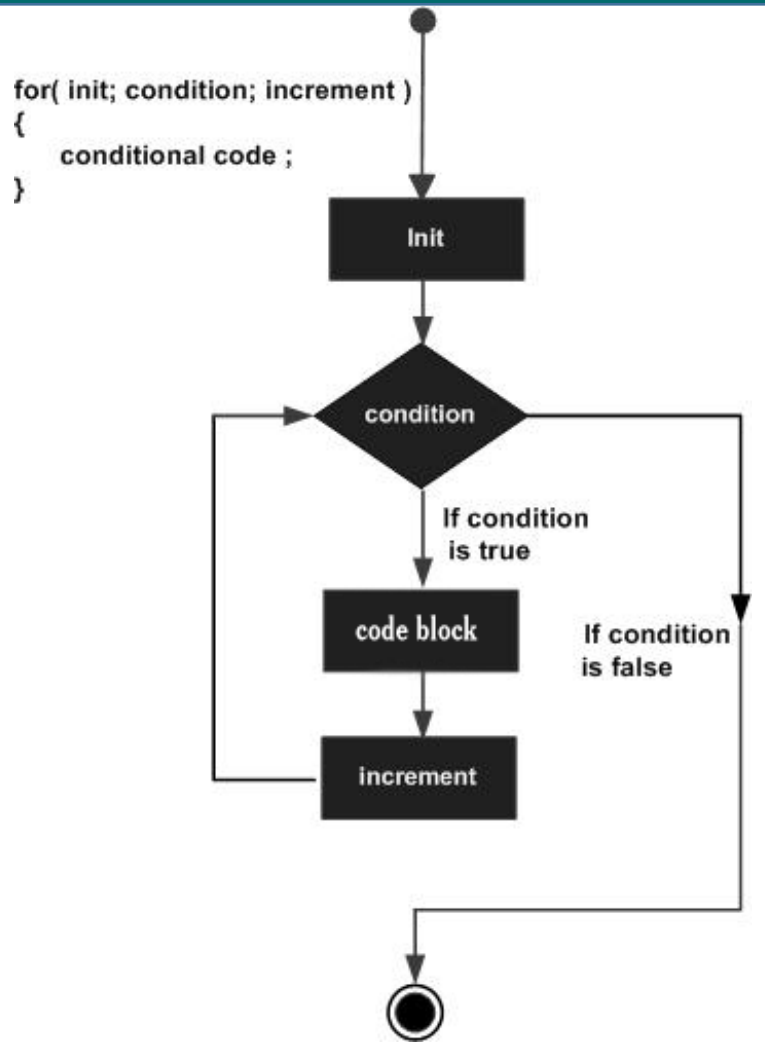


Løkker – While

```
1   using System;
2
3   namespace _2___while
4   {
        0 references
5       class Program
6       {
            0 references
7           static void Main(string[] args)
8           {
9               int a = 10;
10              while (a < 20)
11              {
12                  Console.WriteLine("Værdi af a: {0}", a);
13                  a++;
14              }
15              Console.ReadKey();
16          }
17      }
18  }
```

Løkker – For

- En for-løkke gentager en kode blok indtil en bestemt betingelse er nået.
- I forhold til while løkken har for-løkken indbygget syntaks for initialisering, forøgelse og afprøvning af værdien af en tæller, kaldet loop variablen.
- Fordi der er en specifik placering i loopets syntaks for en tilvækst operation, anvendes increment og decrement operatører ofte som en del af en for-løkke.

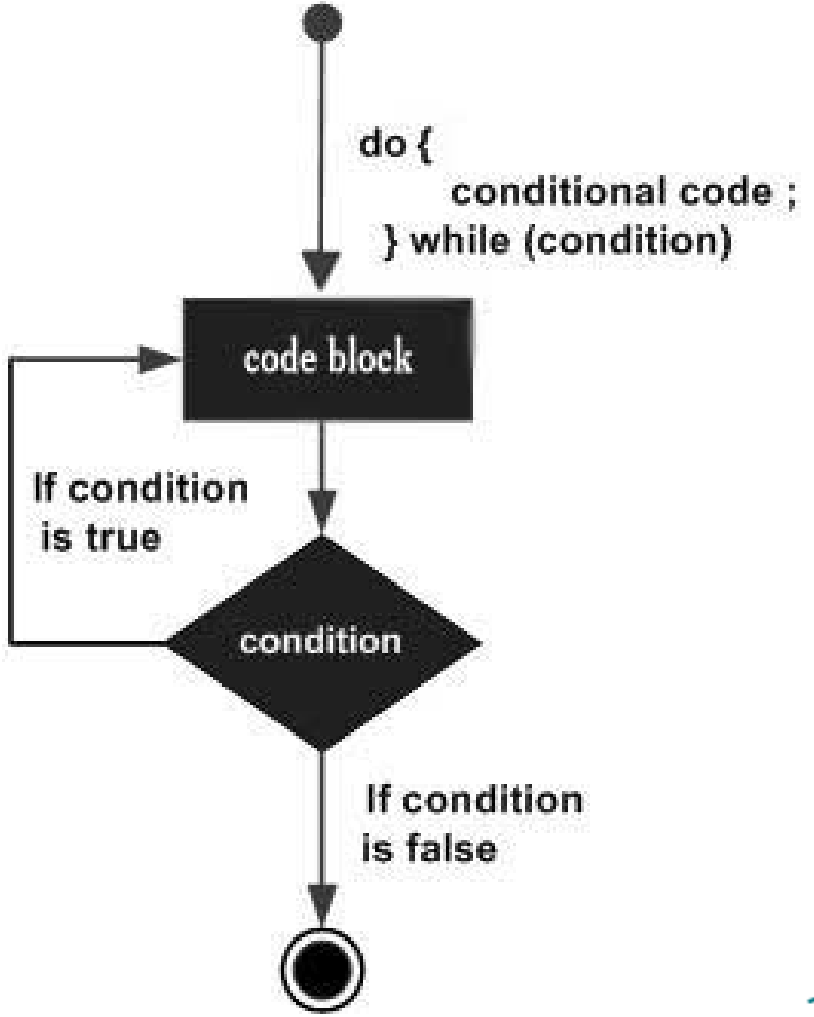


Løkker – For

```
1   using System;
2
3   namespace _3___for
4   {
        0 references
5       class Program
6       {
            0 references
7           static void Main(string[] args)
8           {
9               for(int a = 10; a < 20; a = a+1)
10              {
11                  Console.WriteLine("Værdien af a: {0}", a);
12              }
13              Console.ReadKey();
14          }
15      }
16  }
```

Løkker – Do… while

- Do / while-løkken er meget lig while løkken, bortset fra at do / while løkken foretrækkes, når antallet af gentagelser er fra 1 til n, og n er ikke er kendt når iterationen begynder.
  - Dette opstår ofte når man spørger en bruger efter input.
- Do… while køres altid mindst én gang igennem, da tælleren først tjekkes efter første gennemløb.



Løkker – Do… while

```
1   using System;
2
3   namespace _4___dowhile
4   {
        0 references
5       class Program
6       {
            0 references
7           static void Main(string[] args)
8           {
9               int a = 10;
10              do
11              {
12                  Console.WriteLine("Værdien af a: {0}", a);
13                  a = a + 1;
14              }
15              while (a < 20);
16              Console.ReadKey();
17          }
18      }
19  }
```

for Loop

```
class Program
    {
        static void Main(string[] args)
        {
            for(int i=1; i<10; i++)
            {
                Console.WriteLine("Count is: " + i);
            }
            Console.ReadLine();
        }
    }
```

Random

- Tilfældige tal anvendes i en bred vifte af software applikationer.
- Afhængigt af hvad du bruger tilfældige tal til, skal du bestemme hvilken type du skal bruge.
  - Til en musik jukeboks er nøjagtigheden ikke særlig kritisk. Til noget som et lotteri eller en spilleautomat skal tilfældig talgeneratoren være ekstremt nøjagtig.
- Der er to typer random number generatorer in C#:
  - Pseudo-random numbers (System.Random)
  - Secure random numbers (System.Security.Cryptography.RNGCryptoServiceProvider)

Random

- Den primære forskel er chancen for, at seed værdien anvendt til at gøre randomiseringen muligvis ikke ændrer sig hurtigt og tilfældigt nok.
  - For eksempel er System.Random afhængig af computersystemets ur. Hvis flere tilfældige () objekter blev oprettet på nøjagtig samme tid, kunne de oprette samme sekvens af tilfældige tal.
- Sikre tilfældige tal kaldes "sikre" på grund af mulige sikkerhedsproblemer i svage tilfældige talgeneratorer. Hvis en hacker kunne finde ud af et mønster til dine tilfældige crypto nøgler, kan de muligvis øge deres chancer for hacking.

Vi koder 3D uden modeller

Se unitycsharp.pdf

Kilder

Kilder

Metoder

- https://www.tutorialspoint.com/csharp/csharp_methods.htm
- https://msdn.microsoft.com/da-dk/library/sf0df423.aspx
- https://www.tutorialspoint.com/csharp/csharp_methods.htm
- https://msdn.microsoft.com/da-dk/library/sf0df423.aspx
- https://msdn.microsoft.com/da-dk/library/c3ay4x3d.aspx
- https://www.dotnetperls.com/main
- https://msdn.microsoft.com/da-dk/library/acy3edy3.aspx
- https://youtu.be/QwygwfqOHsI

Kilder

Random

- http://csharpindepth.com/Articles/Chapter12/Random.aspx
- http://www.c-sharpcorner.com/article/yahtzee-program-using-C-Sharp/