Lektion 7

Grundlæggende programmering i VR

Plan for i dag

- Google Cardboard
  - Vi gør vores produkt færdigt
- C# og objekt orienteret programmering
  - Exception handling
  - Validering
- Steder at finde gratis 3D modeller
- Introduktion til Augemented Reality
  - Eksempler på lidt flere enheder
- Unite Austin 2017
- Augmented Reality ved hjælp af Vuforia

Google Cardboard app fra Unity

En code-along video

C# og objekt orienteret programmering

Exception handling og validering

Exception handling

- Problemer opstår så der må gøres noget

Exception handling

- En undtagelse (exception) er et problem, der opstår under udførelsen af et program.
- I C# er en undtagelse er en reaktion på en usædvanlig hændelse, der opstår, mens et program kører, f.eks. et forsøg på at dividere med nul.
- Exceptions sørger for en måde at transfer kontrol fra en del af et program til en anden.
- C# exception handling bygger på fire keywords: **try**, **catch**, **finally** og **throw**.
  - **try**: En try blok identificerer en blok af kode, for hvilken særlige undtagelser er aktiveret. Det efterfølges af en eller flere catch blokke.
  - **catch**: Et program fanger en undtagelse med en exception handler det sted i programmet, hvor du ønsker at håndtere problemet. Catch keyword angiver fangst af en undtagelse.
  - **finally**: Finally blokken anvendes til at udføre et givet sæt udsagn, om en undtagelse er thrown eller ikke thrown. For eksempel, hvis du åbner en fil, skal det være lukket om en undtagelse hæves eller ej.
  - **throw**: Et program kaster en undtagelse, når et problem dukker op. Dette gøres ved hjælp af throw keyword.

Exception handling
Syntaks

- Hvis en blok udløser en exception fanger en metode en exception gennem en kombination af try og catch keywords.
- En try/catch blok placers omrking koden, der kan generere en exception. Kode i en try/catch omtales som protected (beskyttet) kode.
- Du kan liste flere catch statements til at fange forskellige typer af undtagelser i tilfælde af at en blok trigger mere end en enkelt undtagelse i forskellige situationer.

```
try
{
   // statements causing exception
}
catch( ExceptionName e1 )
{
   // error handling code
}
catch( ExceptionName e2 )
{
   // error handling code
}
catch( ExceptionName eN )
{
   // error handling code
}
finally
{
   // statements to be executed
}
```

Exception handling
Klasser

- C# exceptions repræsenteres af classes.
- Exception classes i C# er primært direkte eller indirekte afledt af **System.Exception** klassen.
  - Nogle exception classes afledt af System.Exception klassen er **System.ApplicationException** og **System.SystemException**.
  - **System.ApplicationException** klassen understøtter exceptions genereret af application programmer. Derfor bør exceptions, som defineres af programmørerne, stamme fra denne klasse.
  - **System.SystemException** klassen er base klassen for alle prædefinerede system exceptions.
  - Nogle af de prædefinerede exception klasser afledt af System.SystemException klassen kan ses i skemaet på næste slide.

Exception handling Klasser

Exception Class	Description
System.IO.IOException	Handles I/O errors.
System.IndexOutOfRangeException	Handles errors generated when a method refers to an array index out of range.
System.ArrayTypeMismatchException	Handles errors generated when type is mismatched with the array type.
System.NullReferenceException	Handles errors generated from deferencing a null object.
System.DivideByZeroException	Handles errors generated from dividing a dividend with zero.
System.InvalidCastException	Handles errors generated during typecasting.
System.OutOfMemoryException	Handles errors generated from insufficient free memory.
System.StackOverflowException	Handles errors generated from stack overflow.

Exception handling
Håndtering

Håndtering af exceptions

- C # giver en struktureret løsning på håndtering af exceptions i form af try og catch blokke. Ved hjælp af disse blokke bliver kerne-program statements adskilt fra fejl håndterings statements.
- Disse fejl håndterings blokke implementers med **try**, **catch**, og **finally** keywords.

Exception handling
Håndtering

```
1   using System;
2   namespace ErrorHandlingApplication
3   {
        3 references
4       class DivNumbers
5       {
6           int result;
            1 reference
7           DivNumbers()
8           {
9               result = 0;
10          }
            1 reference
11          public void division(int num1, int num2)
12          {
13              try
14              {
15                  result = num1 / num2;
16              }
17              catch (DivideByZeroException e)
18              {
19                  Console.WriteLine("Exception caught: {0}", e);
20              }
21              finally
22              {
23                  Console.WriteLine("Result: {0}", result);
24              }
25          }
            0 references
26          static void Main(string[] args)
27          {
28              DivNumbers d = new DivNumbers();
29              d.division(25, 0);
30              Console.ReadKey();
31          }
32      }
33  }
```

Exception handling
Brugerdefinerede

- Man kan også definer ens egne exceptions.
- Bruger definerede exception klasser er afledt af **Exception** klassen.

```
1   using System;
2   namespace UserDefinedException
3   {
        0 references
4       class TestTemperature
5       {
            0 references
6           static void Main(string[] args)
7           {
8               Temperature temp = new Temperature();
9               try
10              {
11                  temp.showTemp();
12              }
13              catch (TempIsZeroException e)
14              {
15                  Console.WriteLine("TempIsZeroException: {0}", e.Message);
16              }
17              Console.ReadKey();
18          }
19      }
20  }
21
        3 references
22  public class TempIsZeroException : Exception
```

```
23  {
        1 reference
24      public TempIsZeroException(string message) : base(message)
25      {
26      }
27  }
28
        2 references
29  public class Temperature
30  {
31      int temperature = 0;
            1 reference
32      public void showTemp()
33      {
34          if (temperature == 0)
35          {
36              throw (new TempIsZeroException("Zero Temperature found"));
37          }
38          else
39          {
40              Console.WriteLine("Temperature: {0}", temperature);
41          }
42      }
43  }
```

Validation

- Tjek af programmet

Validation

- Datavalidering er test-værdier introduceret til en app (via en tjeneste, fil eller indtastning af data) mod forventede værdier og ranges.
- Man gør det for at:
  - Undgå overflow.
  - Undgå forkerte resultater.
  - Undgå uønskede bivirkninger
  - Vejlede systemer eller brugere.
  - Forebyg sikkerheds indtrængen.
- Compileren validerer, at objekttypen er korrekt, den validerer ikke objektets værdi.

Gratis 3D modeller

Udover den indbyggede butik

Her er nogle steder I kan få gratis 3D modeller så I lettere kan komme i gang med jeres opgaver

- https://www.thepixellab.net/7-sites-for-free-3d-models
- https://www.hongkiat.com/blog/60-excellent-free-3d-model-websites/
- https://free3d.com/3d-models/fbx
- http://www.creativebloq.com/3d/free-3d-models-10121127

Eksempler på AR i praksis

Forskellige apps på forskellige enheder vises frem og prøves af

Unite Austin 2017

Introduktion til integration af Unity 2017 og Vuforia

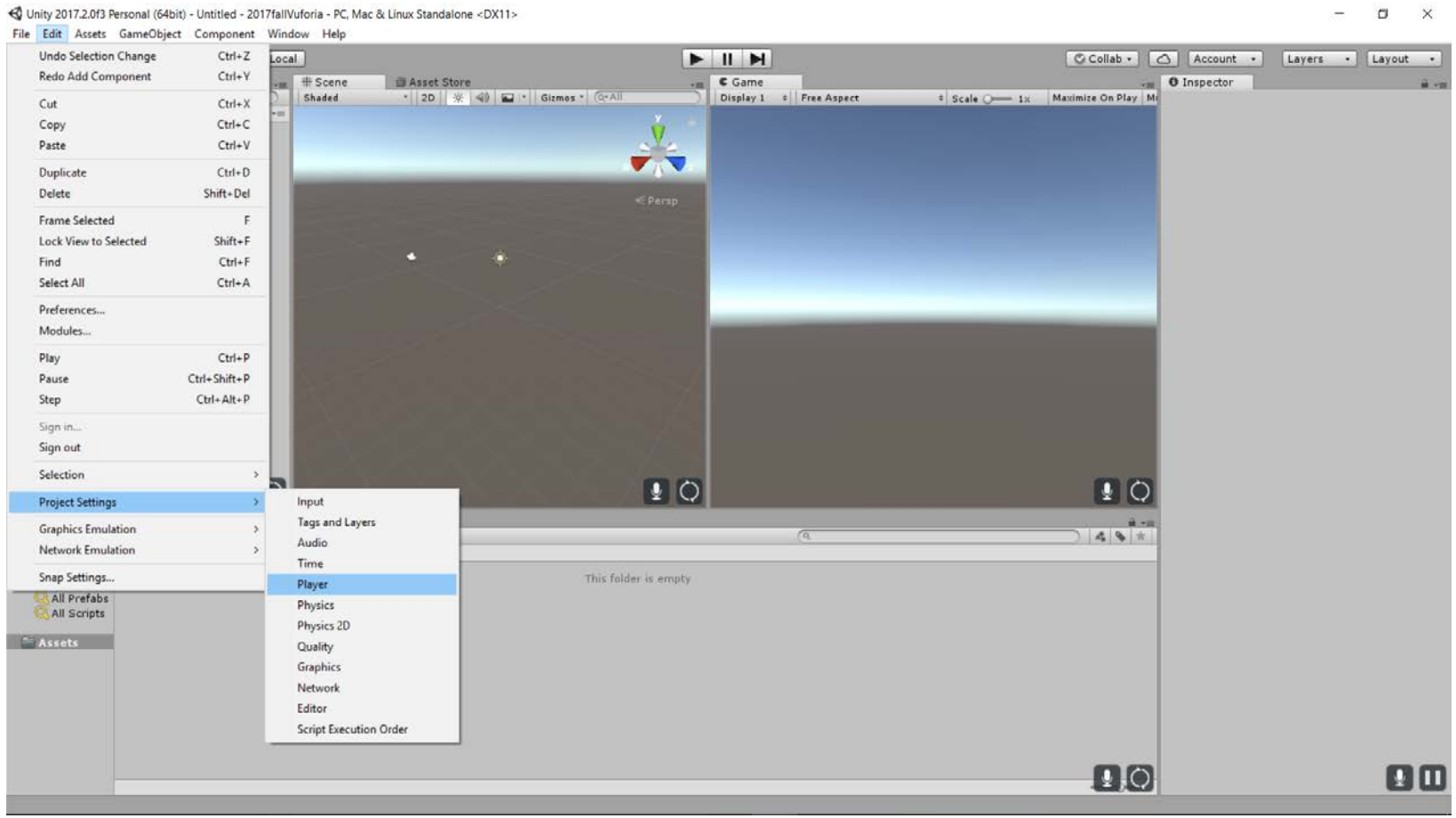

unity
vuforia
Unite
Austin
2017

Augmented Reality ved hjælp af Vuforia

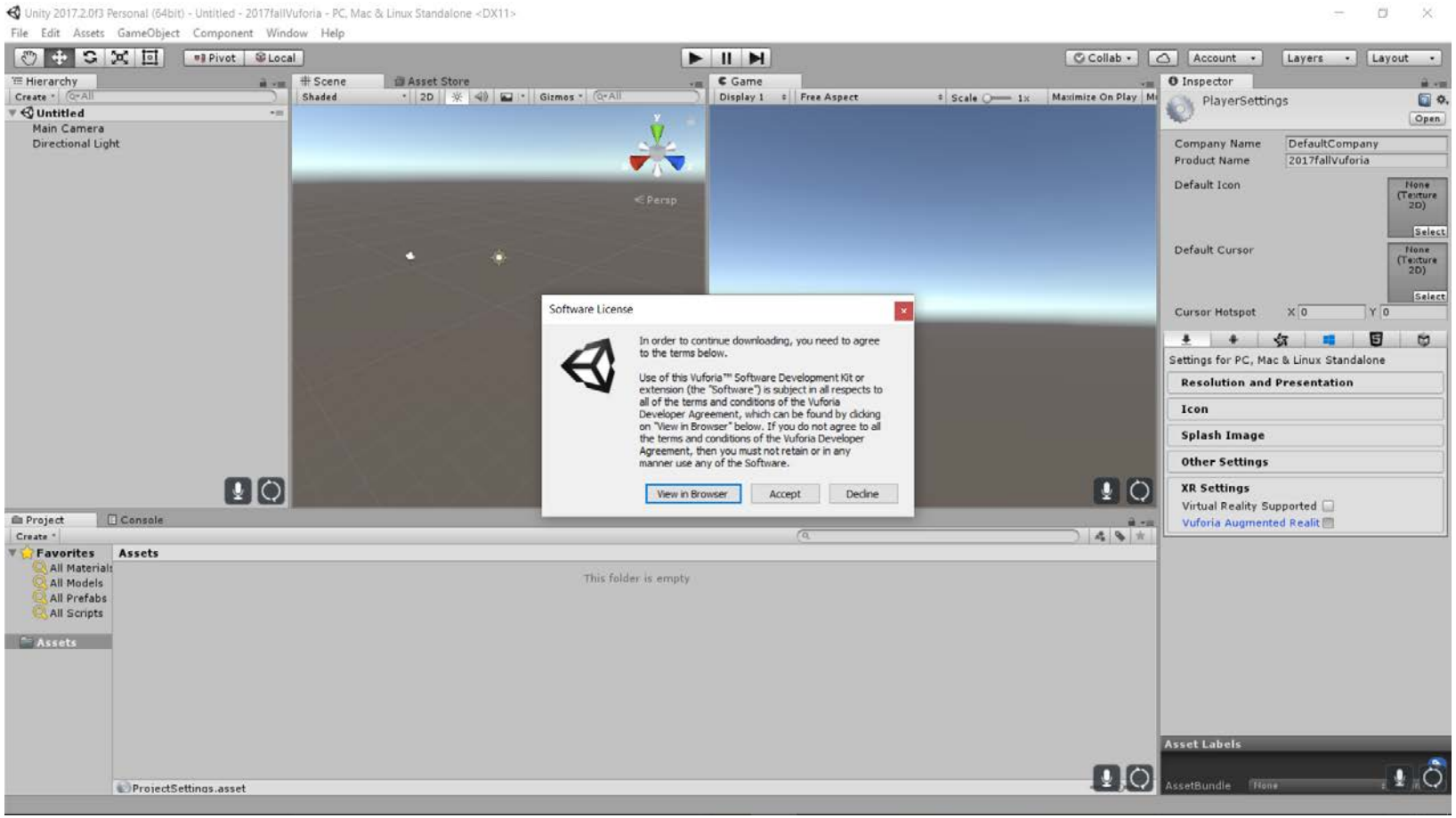
En code-along video

Lav et nyt Unity 3D projekt

Gå ind i Player settings…

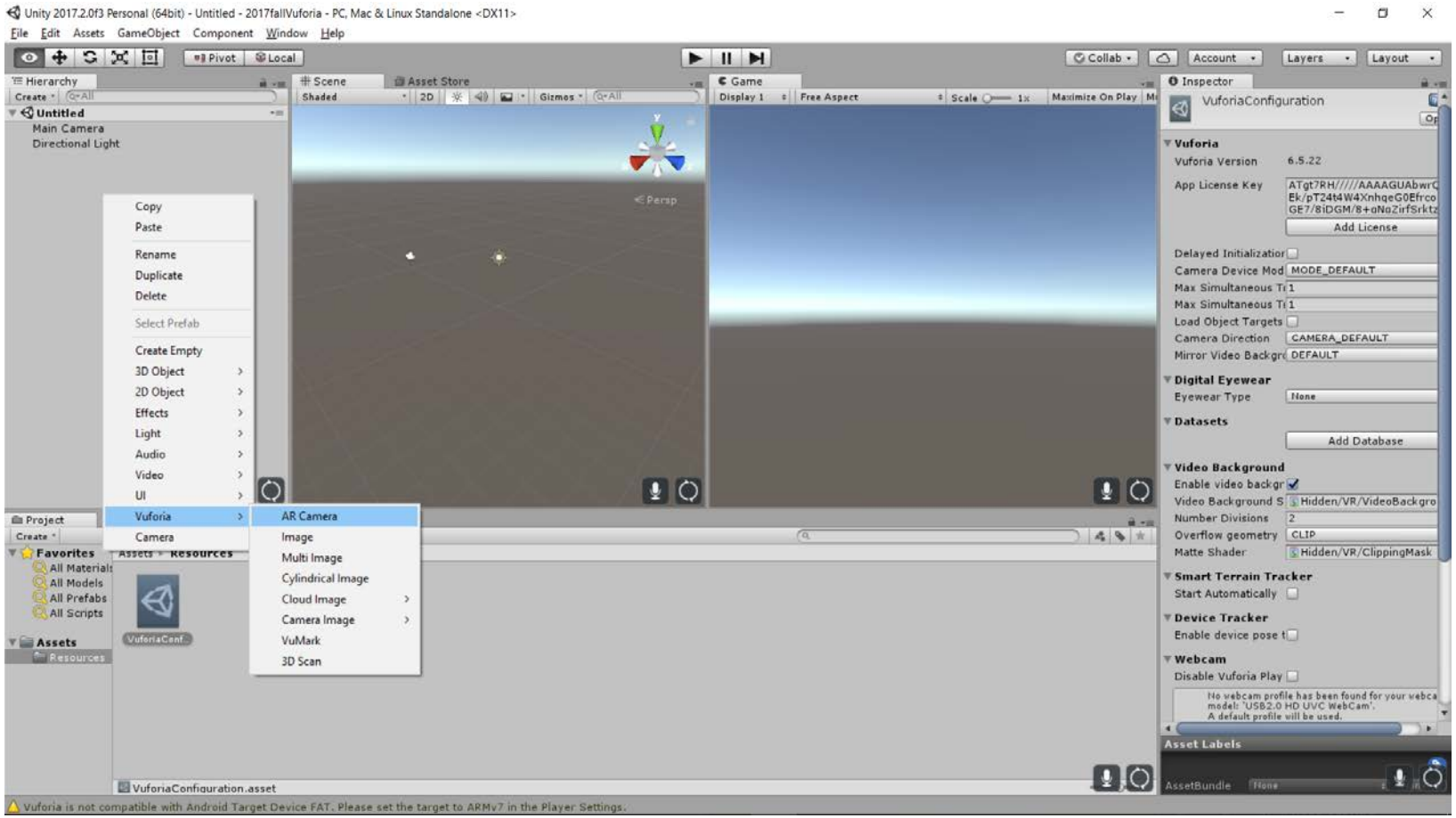


… og aktiver Vuforia

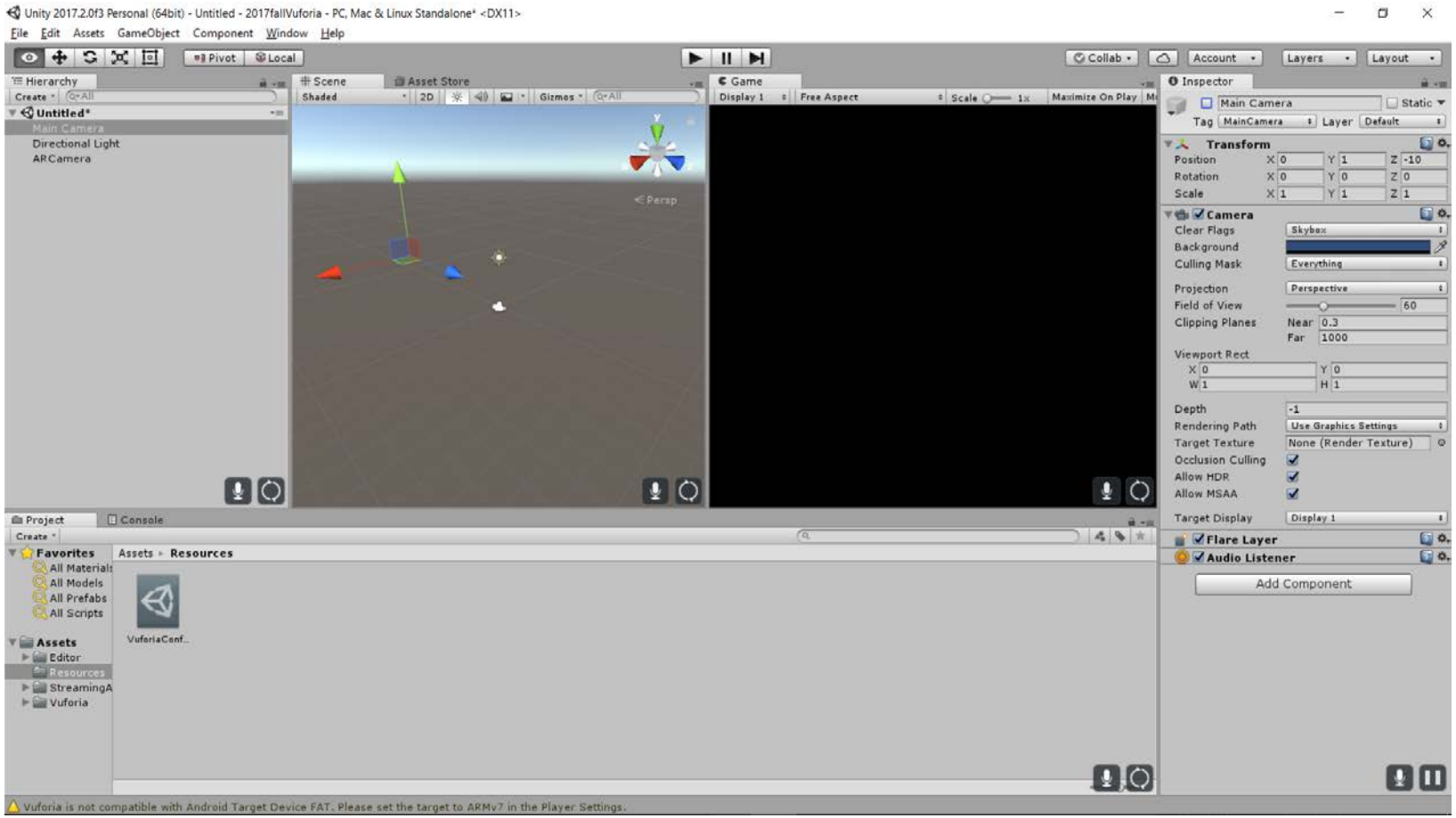


Aktiver Vuforia med en gratis licens

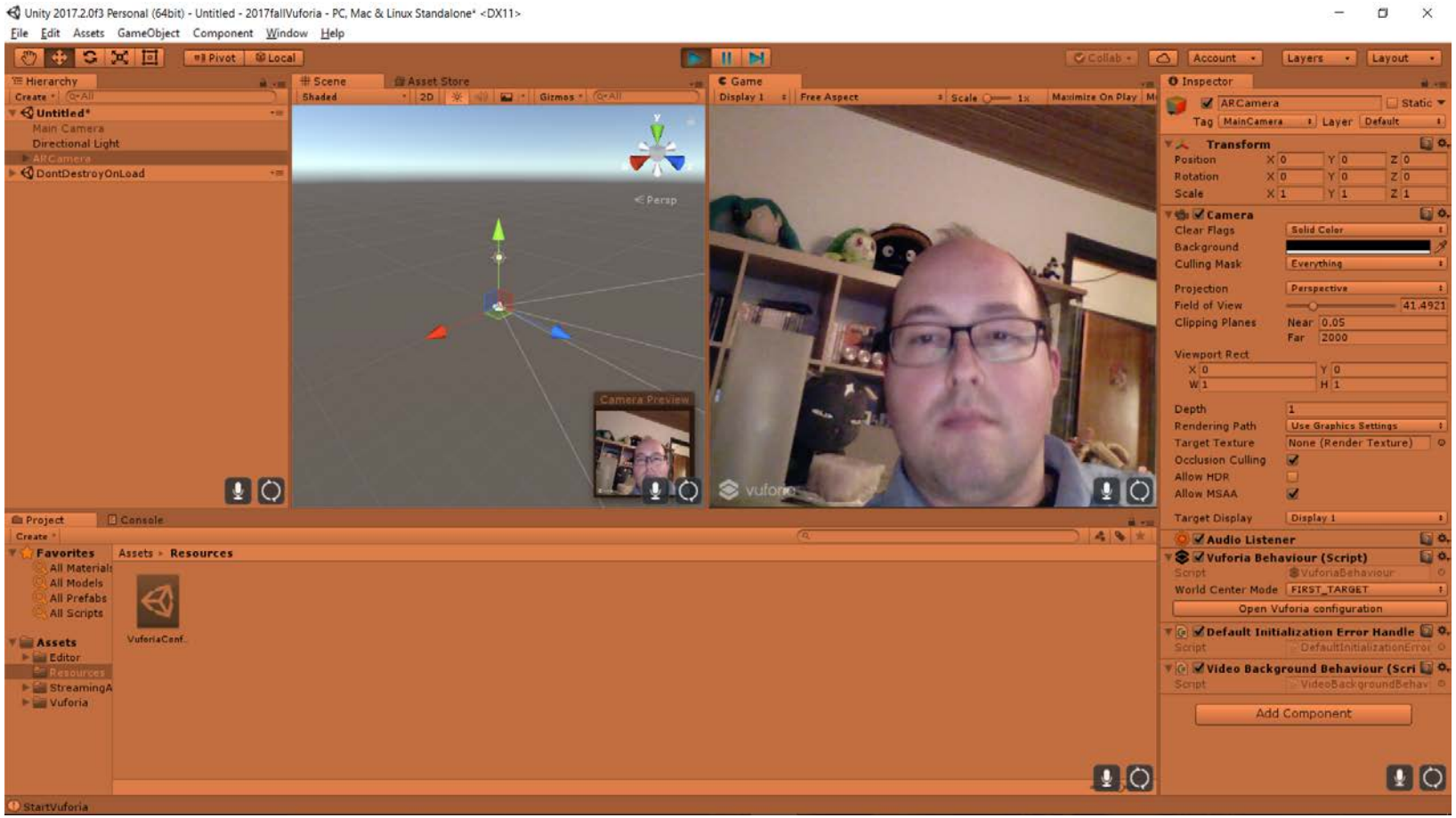
Føj AR Camera til som nyt kamera…



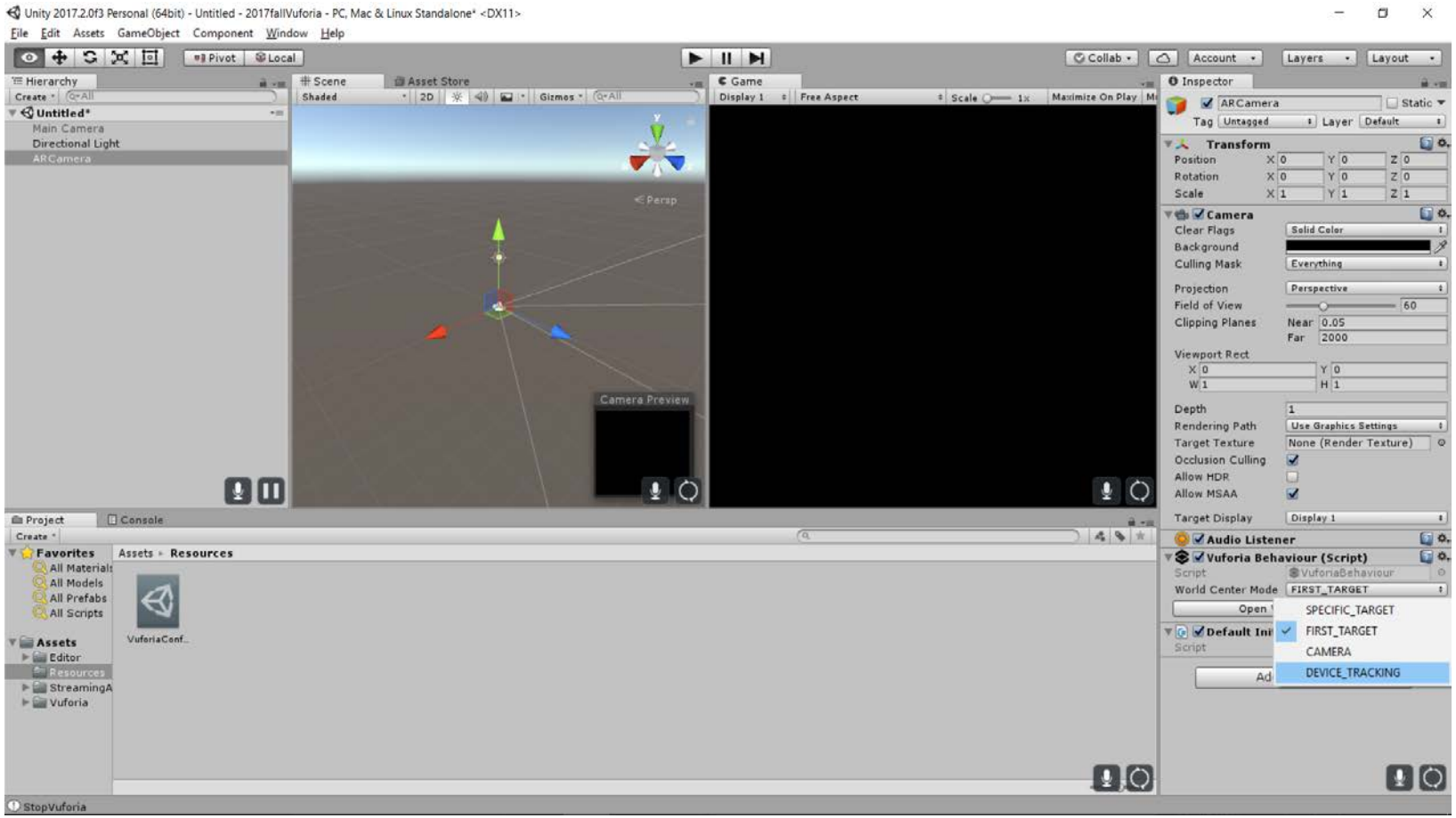
… og deaktiver ellers slet Main Camera



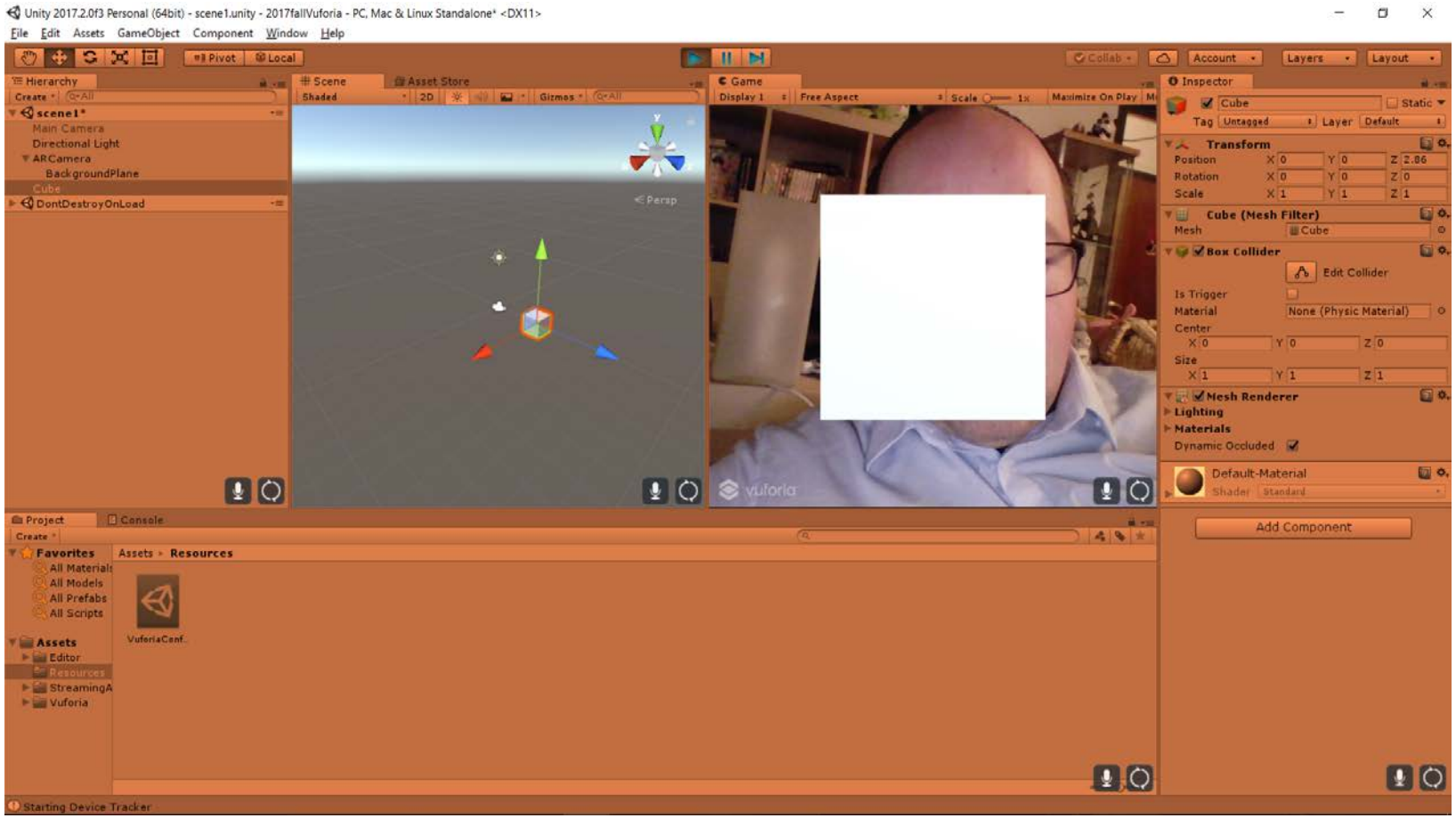
Tryk Play for at tjekke at kameraet virker



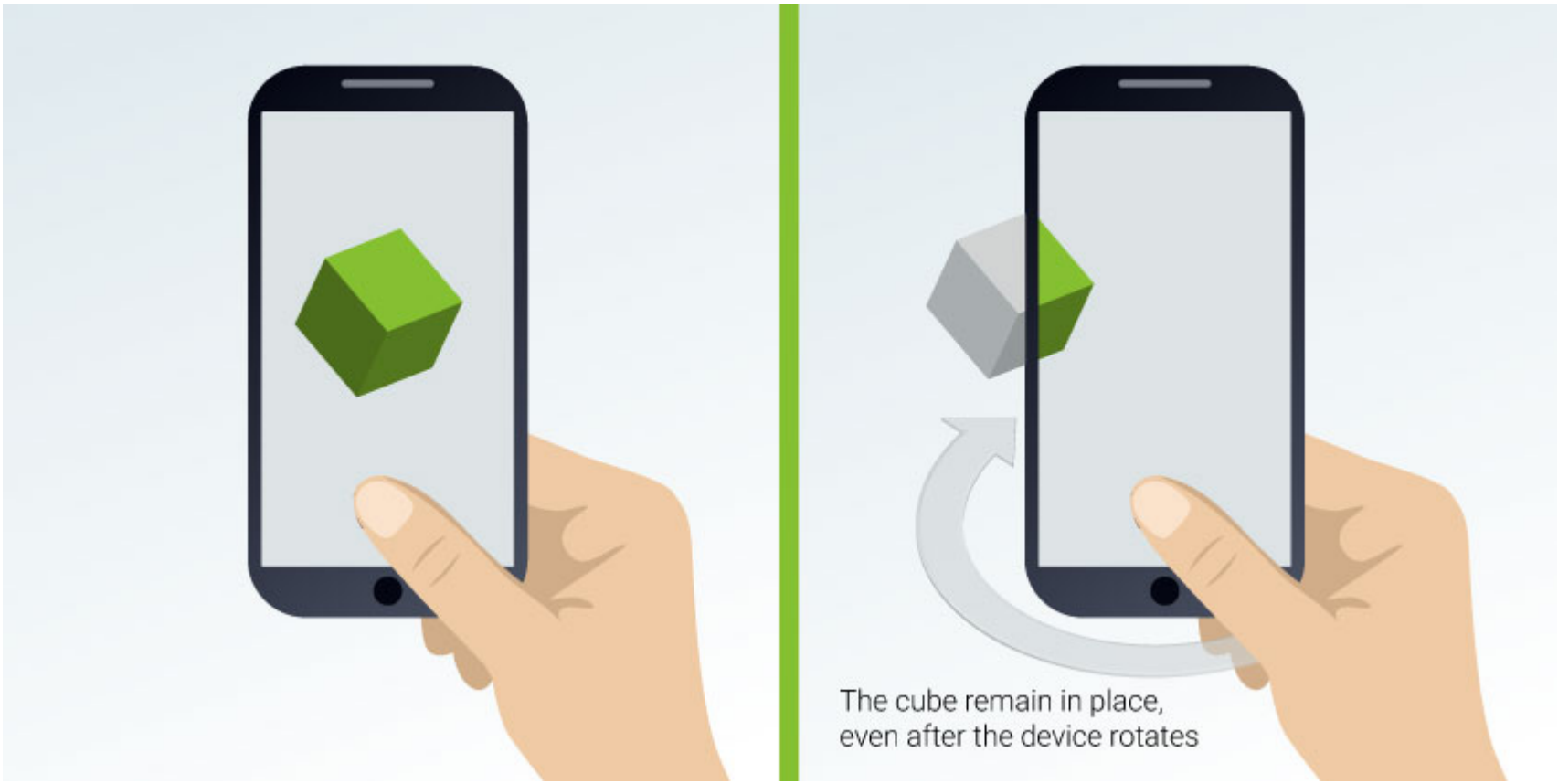
I AR Camera: Skift hvordan systemet finder verdens midte til DEVICE_TRACKING



Lav en kube for at se om alt virker



Som det er lige nu risikerer man at skulle vende sig for at finde kuben da den har en fast position



- Da verdens midte er defineret af enhedens placering kan vi ikke vide den faktiske position på elementerne.
- Dette er fordi brugeren skal kunne starte med sin enhed i hvilken som helst orientering, og fordi rotation måles forskelligt fra enhed til enhed.
- For at sikre, at AR-enhederne starter i forhold til brugeren, er det nemmeste at vente på, at Vuforia definerer verdens midte og finde ARCamera-rotationen og derefter arrangere startstedet for elementer i overensstemmelse med dets orientering.

- Vi laver en Spawn Manager til at definere placeringen af kubernes opståen.
- Manageren vil definere sin position i henhold til ARCameras rotation en venter, indtil rotationen er indstillet, og flytter derefter 10 enheder til kameraets forside.

- Lav to tomme game objects
- Højreklik på den ene og kald den _SpawnController
- Skift navnet på den anden til _GameManager
- Lav en ny mappe i Assets der hedder Scripts
- Under Scripts lavet vi et nyt C# script kaldet SpawnScript
- Føj SpawnScript til _SpawnController

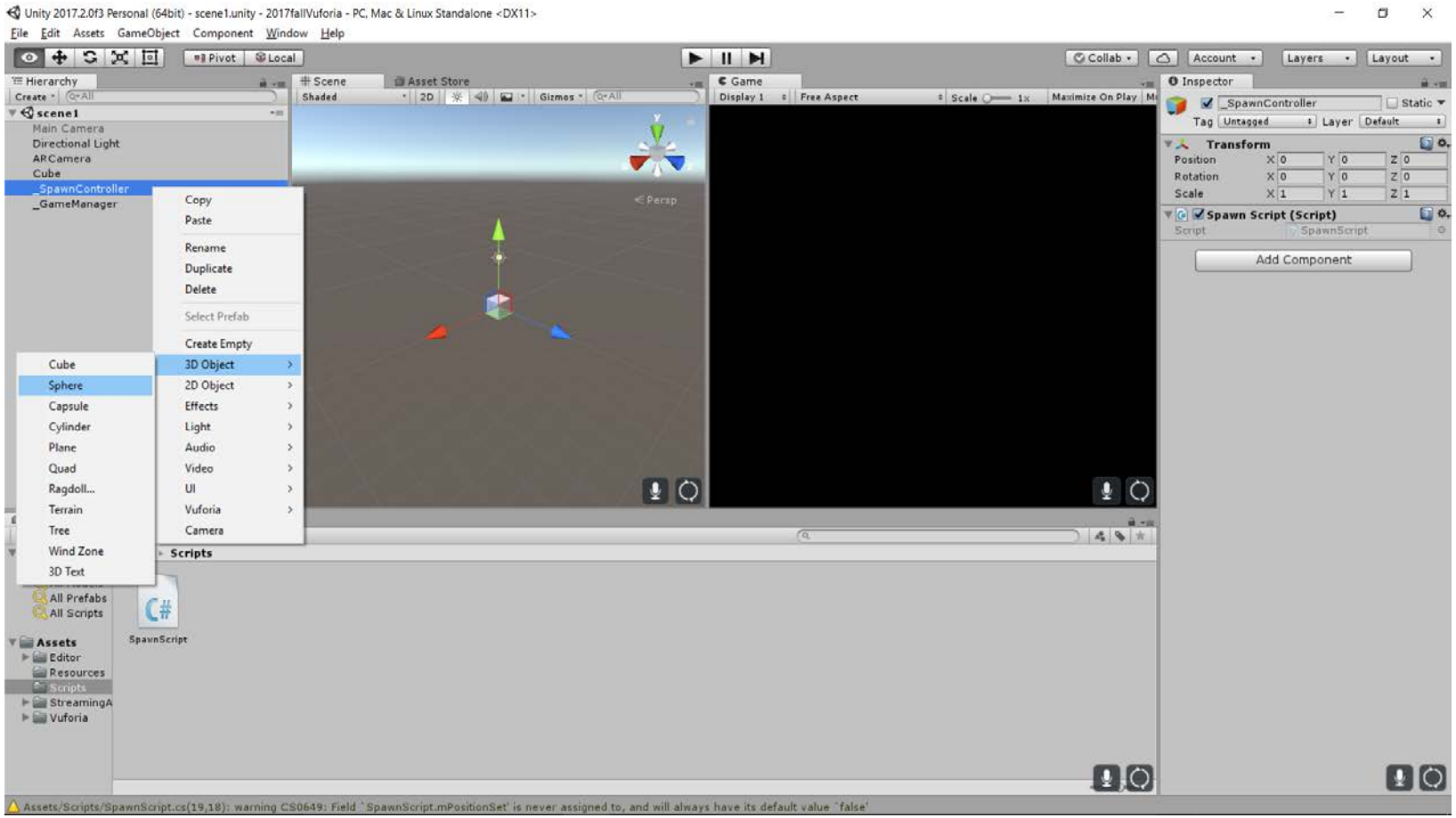
Rediger SpawnScript

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4   using Vuforia;
5
6   public class SpawnScript : MonoBehaviour {
7
8       // Define the position if the object according to ARCamera position
9       private bool SetPosition()
10      {
11          // get the camera position
12          Transform cam = Camera.main.transform;
13
14          // set the position 10 units forward from the camera position
15          transform.position = cam.forward * 10;
16          return true;
17      }
18
19      private bool mPositionSet;
```

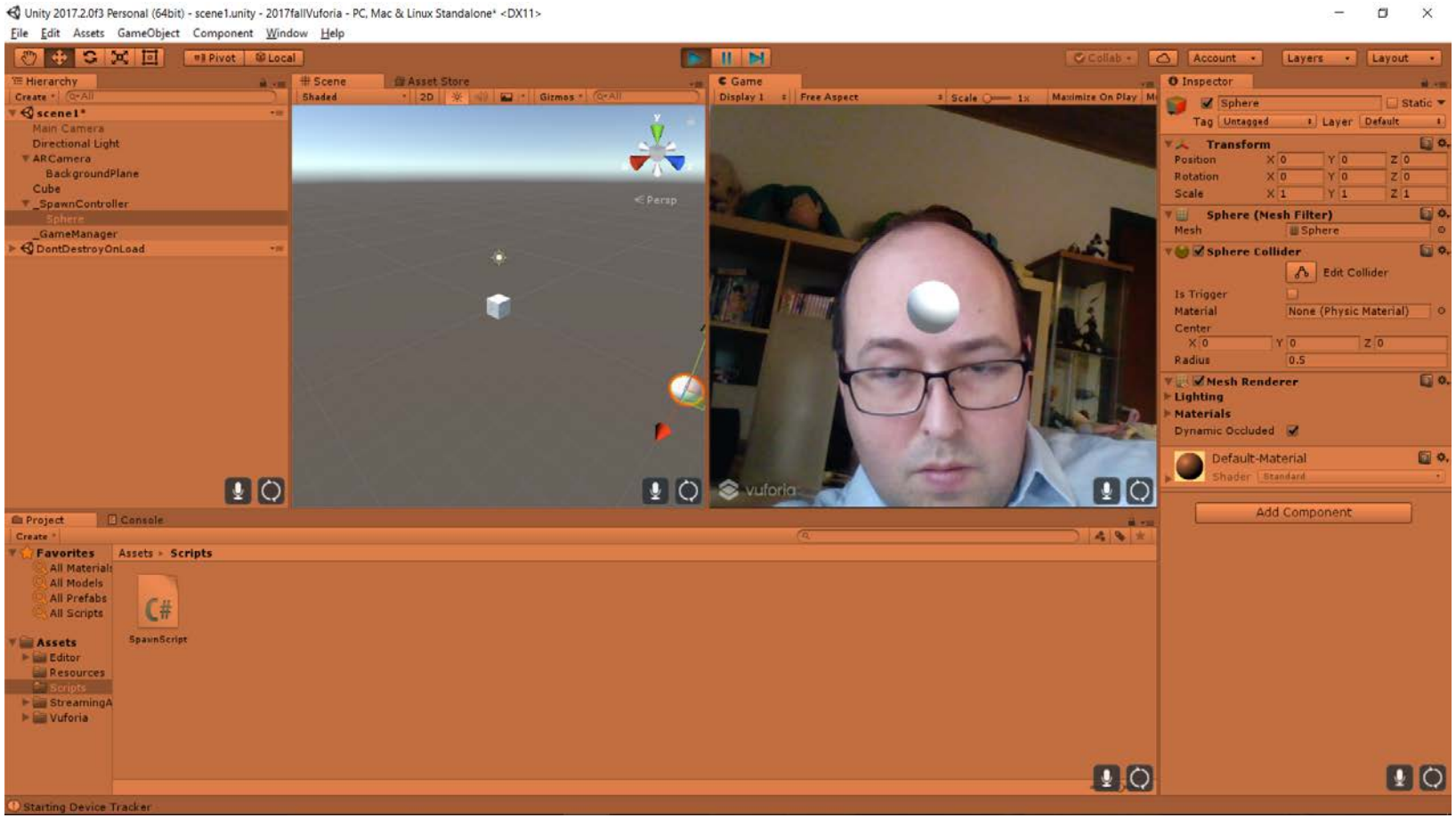
Rediger SpawnScript

```
20
21      // Use this for initialization
22      void Start () {
23          // Defining the Spawning Position
24          StartCoroutine(ChangePosition());
25      }
26
27      // We'll use a Coroutine to give a little delay before setting the position
28      private IEnumerator ChangePosition()
29      {
30          yield return new WaitForSeconds(0.2f);
31          // Define the Spawn position only once
32          if (!mPositionSet)
33          {
34              // change the position only if Vuforia is active
35              if (VuforiaBehaviour.Instance.enabled)
36                  SetPosition();
37          }
38      }
```

Put en Sphere under vores _SpawnController



Selvom de har samme koordinater vil bolden være et andet sted end kuben når app'en køres



- Nu vi har set vores app køre vil vi lave lidt mere liv. Vores SpawnScript vil få _SpawnController skal spawne kuber i forskellige størrelser og positioner i forhold til _SpawnController.

Rediger SpawnScript

```
5
6   public class SpawnScript : MonoBehaviour {
7       // Cube element to spawn
8       public GameObject mCubeObj;
9
10      // Qtd of Cubes to be Spawned
11      public int mTotalCubes = 10;
12
13      // Time to spawn the Cubes
14      public float mTimeToSpawn = 1f;
15
16      // hold all cubes on stage
17      private GameObject[] mCubes;
18
19      // define if position was set
20      private bool mPositionSet;
21
22      // Define the position if the object according to ARCamera position
23      private bool SetPosition()
```

Rediger SpawnScript

```
22      // Define the position of the object according to ARCamera position
23      private bool SetPosition()
24      {
25          // get the camera position
26          Transform cam = Camera.main.transform;
27
28          // set the position 10 units forward from the camera position
29          transform.position = cam.forward * 10;
30          return true;
31      }
32
33      // Use this for initialization
34      void Start () {
35          // Initializing spawning loop
36          StartCoroutine(SpawnLoop());
37
38          // Initialize Cubes array according to
39          // the desired quantity
40          mCubes = new GameObject[mTotalCubes];
```

Rediger SpawnScript

```
40          mCubes = new GameObject[mTotalCubes];
41      }
42      // Loop Spawning cube elements
43      // Loop Spawning cube elements
44      private IEnumerator SpawnLoop()
45      {
46          // Defining the Spawning Position
47          StartCoroutine(SpawnLoop());
48
49          yield return new WaitForSeconds(0.2f);
50
51          // Spawning the elements
52          int i = 0;
53          while (i <= (mTotalCubes - 1))
54          {
55
56              mCubes[i] = SpawnElement();
57              i++;
58              yield return new WaitForSeconds(Random.Range(mTimeToSpawn, mTimeToSpawn * 3));
```

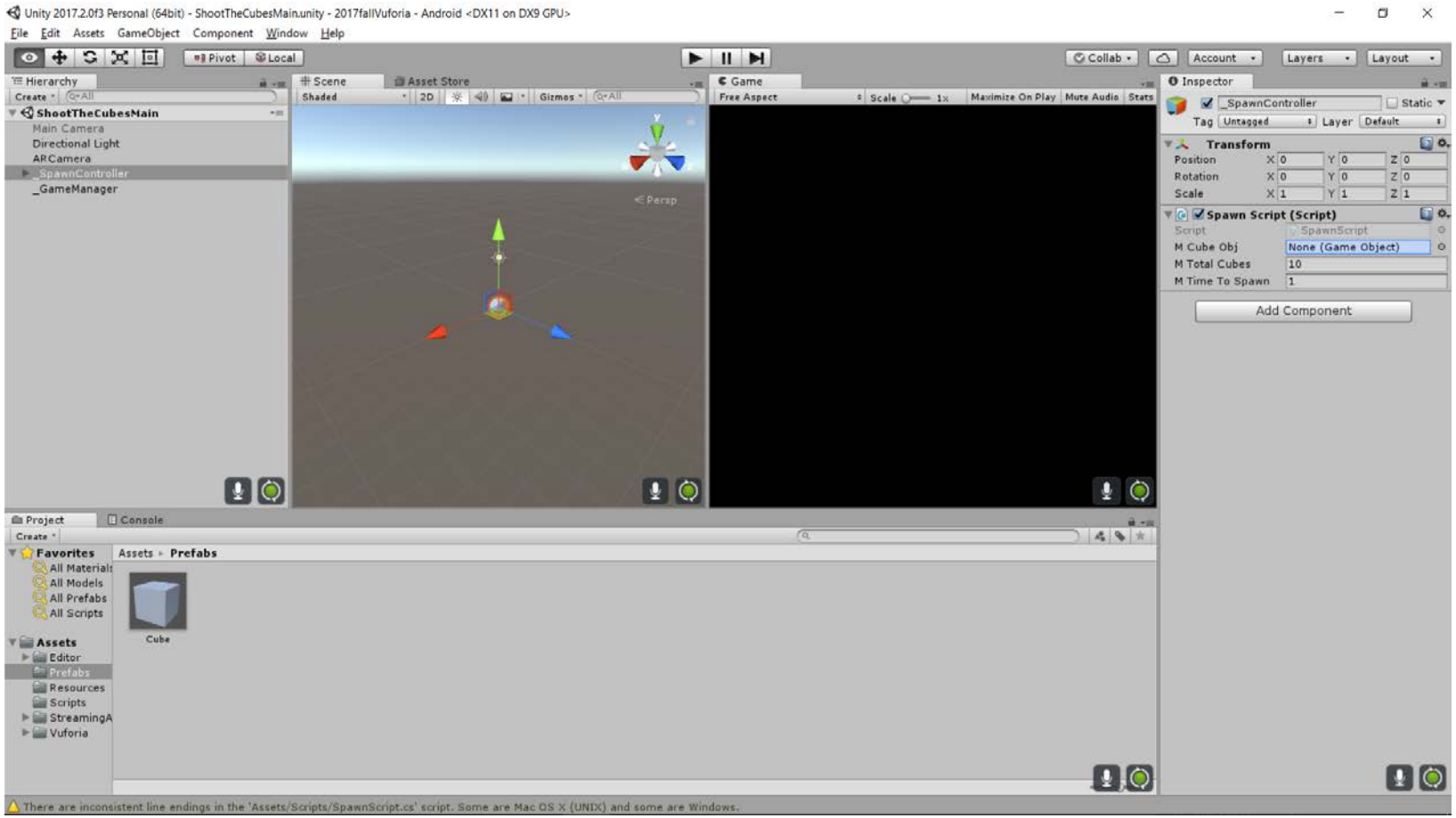
Rediger SpawnScript

```
58              yield return new WaitForSeconds(Random.Range(mTimeToSpawn, mTimeToSpawn * 3));
59          }
60      }
61
62      // Spawn a cube
63      private GameObject SpawnElement()
64      {
65          // spawn the element on a random position, inside a imaginary sphere
66          GameObject cube = Instantiate(mCubeObj, (Random.insideUnitSphere * 4) + transform.position, transform.rotation) as GameObject;
67          // define a random scale for the cube
68          float scale = Random.Range(0.5f, 2f);
69          // change the cube scale
70          cube.transform.localScale = new Vector3(scale, scale, scale);
71          return cube;
72      }
73
74      // Update is called once per frame
75      void Update () {
76      }
77  }
78
```

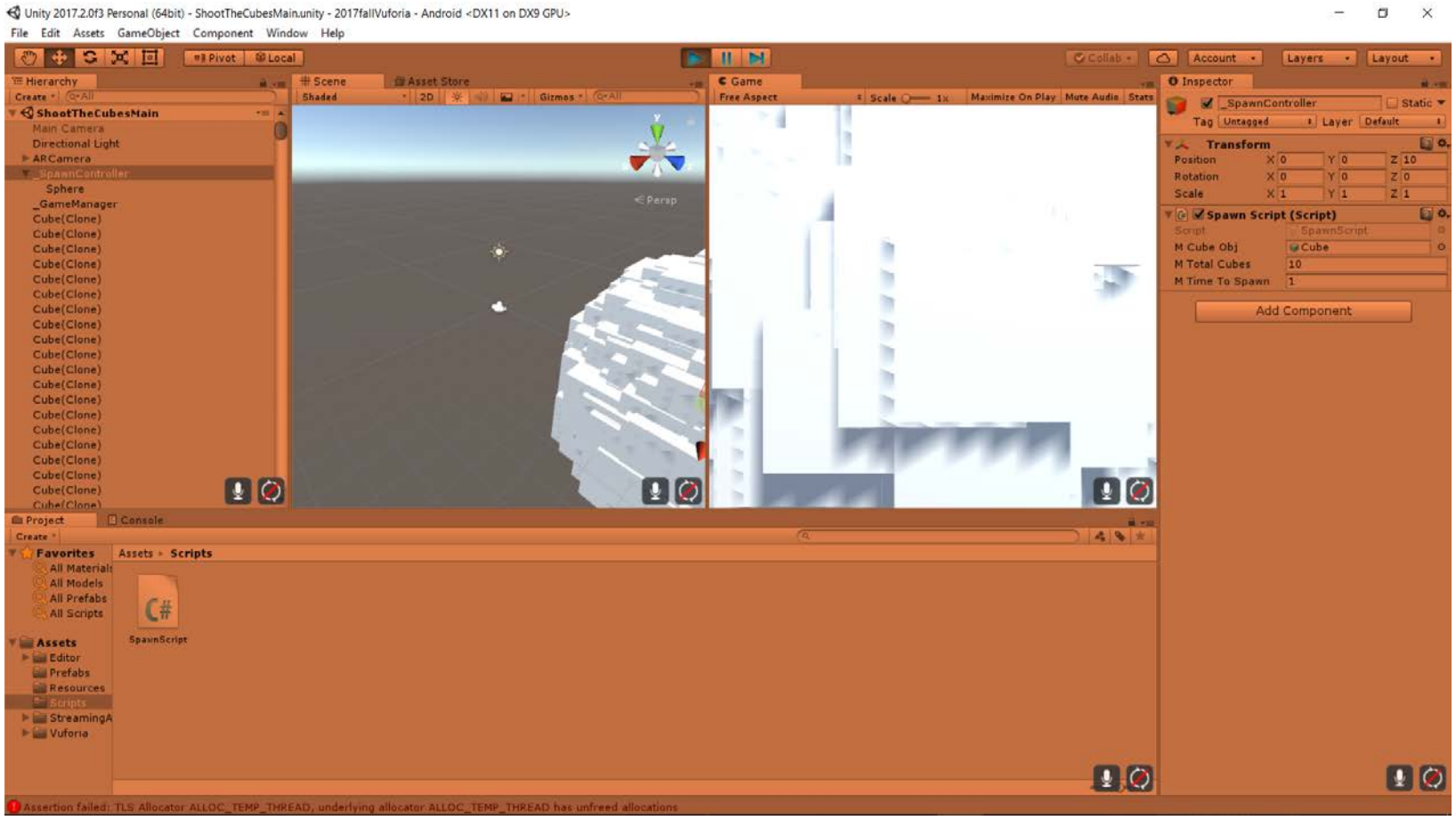
Lav en mappe der hedder Prefabs i Assets

Sikr at kuben er 1:1:1 på alle akser, træk den over i Prefab mappen og slet den fra hierakiet

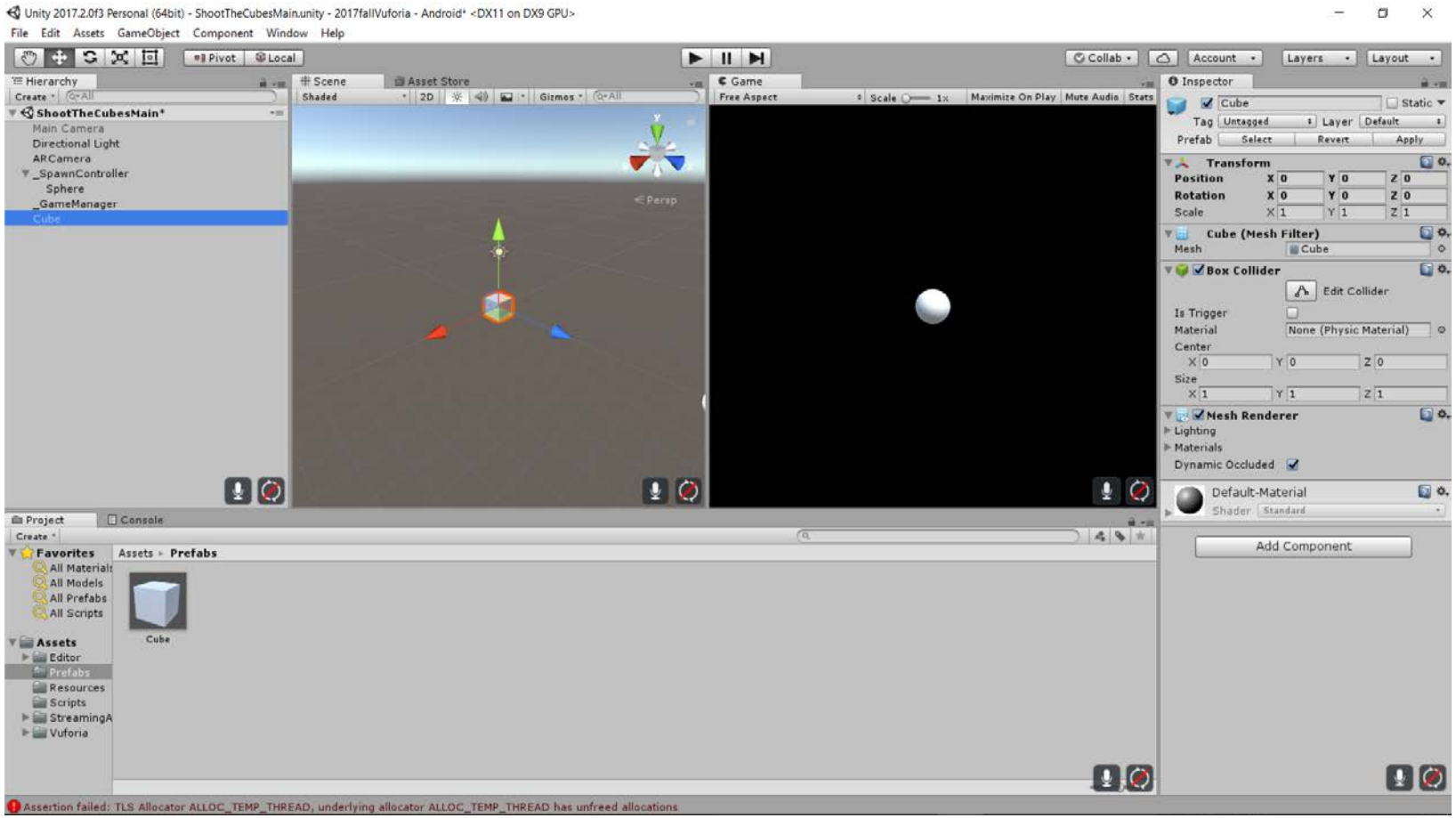
Åbn _SpawnController og træk kuben fra Prefabs til M Cube Obj feltet



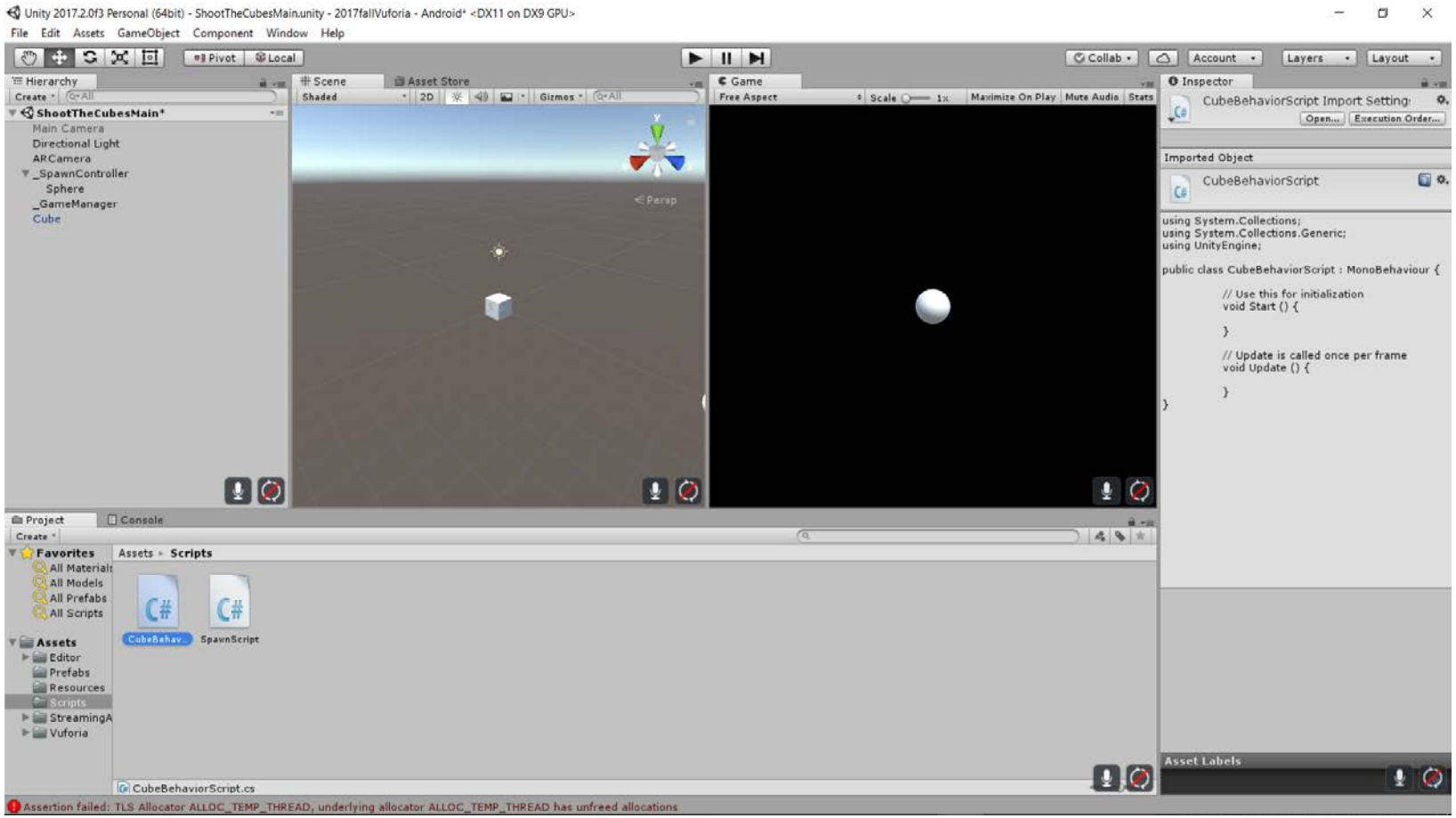
Vi har nu firkanter foran os



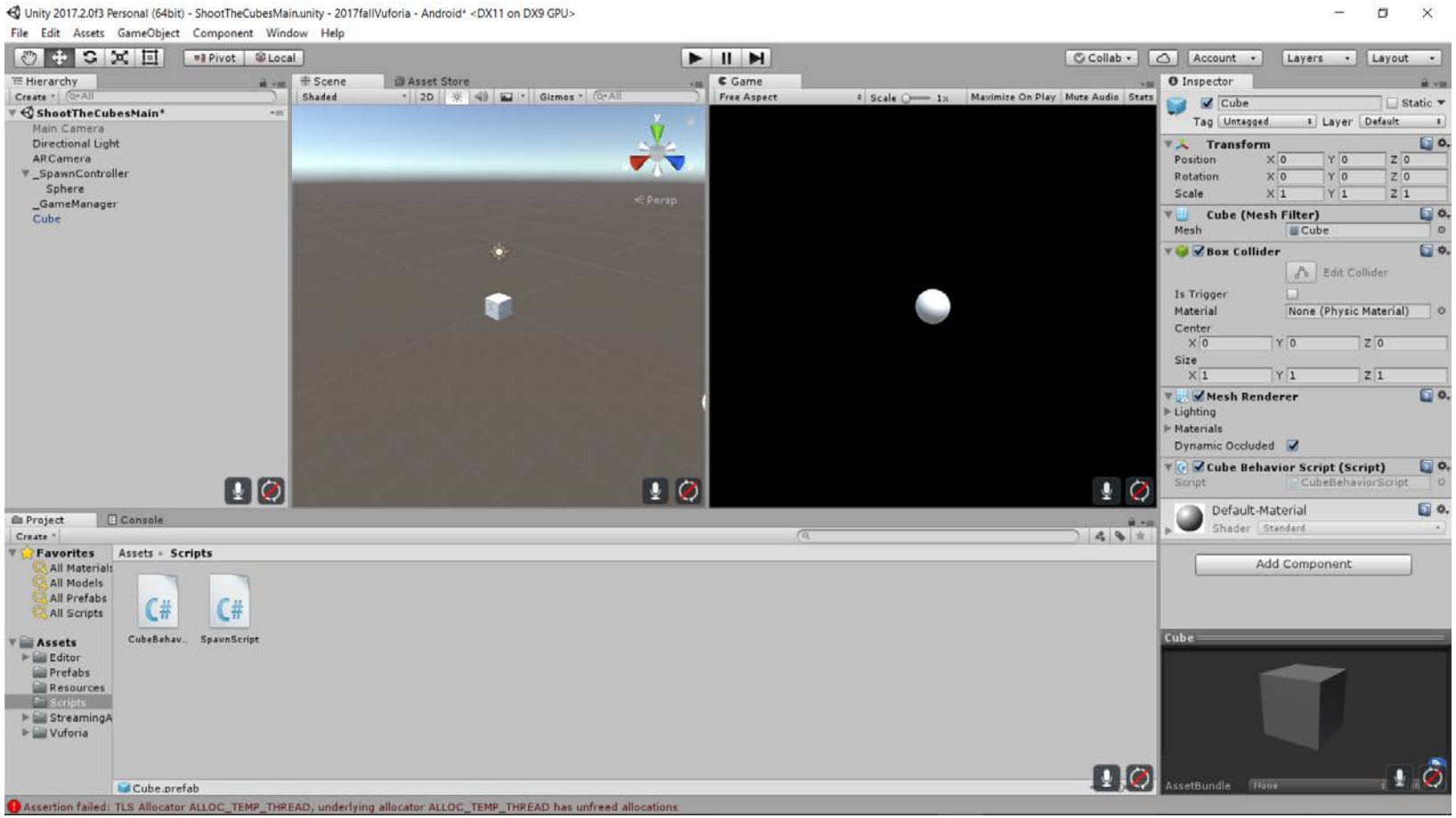
Træk kuben fra Prefabs til hierakiet igen



Lav et nyt script kaldet CubeBehaviorScript



Føj CubeBehaviorScript til vores kube prefab



Rediger CubeBehaviorScript

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class CubeBehaviorScript : MonoBehaviour
5   {
6       // Cube's Max/Min scale
7       public float mScaleMax = 2f;
8       public float mScaleMin = 0.5f;
9
10      // Orbit max Speed
11      public float mOrbitMaxSpeed = 30f;
12
13      // Orbit speed
14      private float mOrbitSpeed;
15
16      // Anchor point for the Cube to rotate around
17      private Transform mOrbitAnchor;
18
19      // Orbit direction
20      private Vector3 mOrbitDirection;
21
22      // Max Cube Scale
23      private Vector3 mCubeMaxScale;
24
25      // Growing Speed
26      public float mGrowingSpeed = 10f;
27      private bool mIsCubeScaled = false;
28
29      void Start()
```

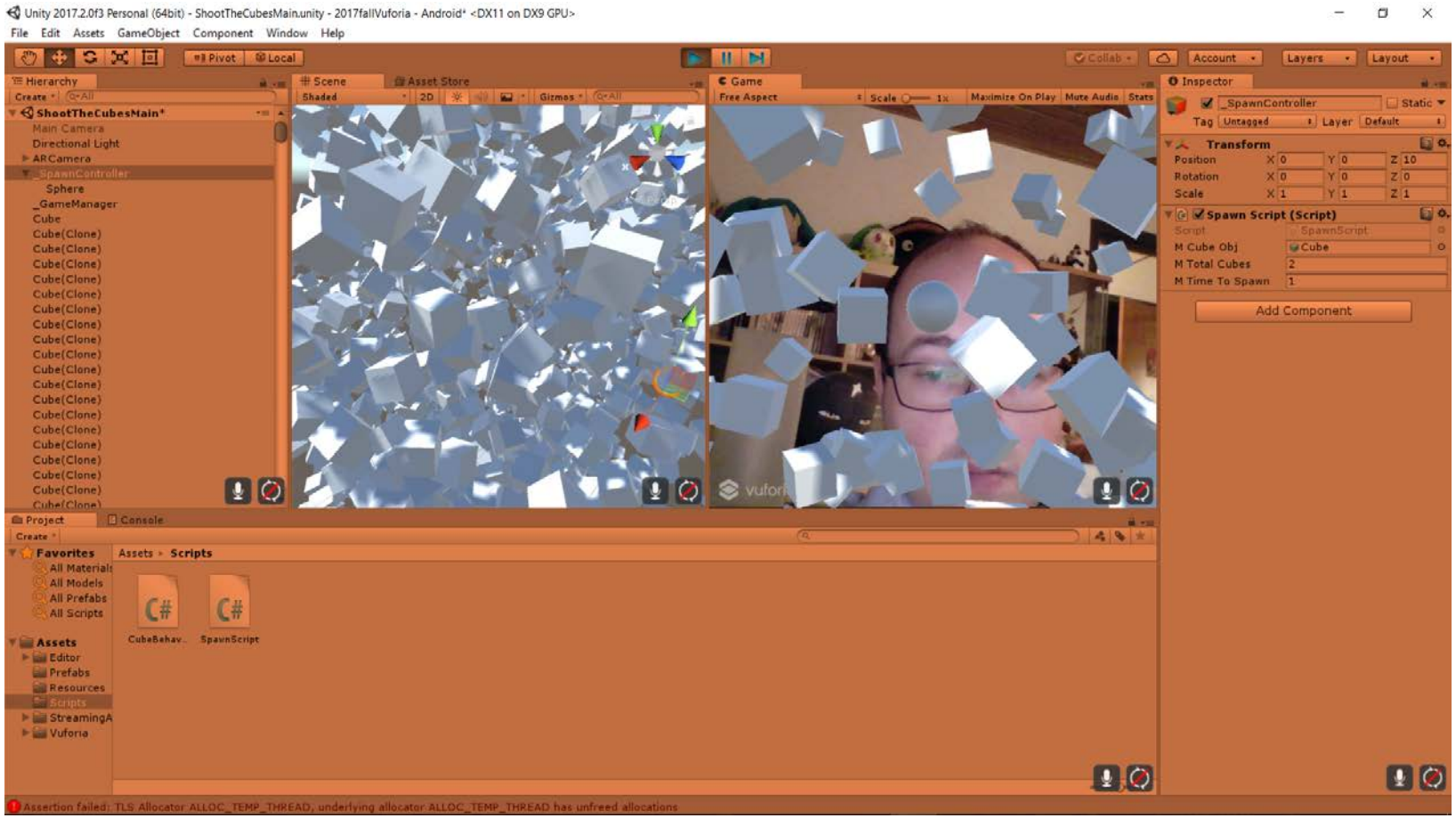
Rediger CubeBehaviorScript

```
30      {
31          CubeSettings();
32      }
33
34      // Set initial cube settings
35      private void CubeSettings()
36      {
37          // defining the anchor point as the main camera
38          mOrbitAnchor = Camera.main.transform;
39
40          // defining the orbit direction
41          float x = Random.Range(-1f, 1f);
42          float y = Random.Range(-1f, 1f);
43          float z = Random.Range(-1f, 1f);
44          mOrbitDirection = new Vector3(x, y, z);
45
46          // defining speed
47          mOrbitSpeed = Random.Range(5f, mOrbitMaxSpeed);
48
49          // defining scale
50          float scale = Random.Range(mScaleMin, mScaleMax);
51          mCubeMaxScale = new Vector3(scale, scale, scale);
52
53          // set cube scale to 0, to grow it lates
54          transform.localScale = Vector3.zero;
55      }
56
57      // Update is called once per frame
58      void Update()
```

Rediger CubeBehaviorScript

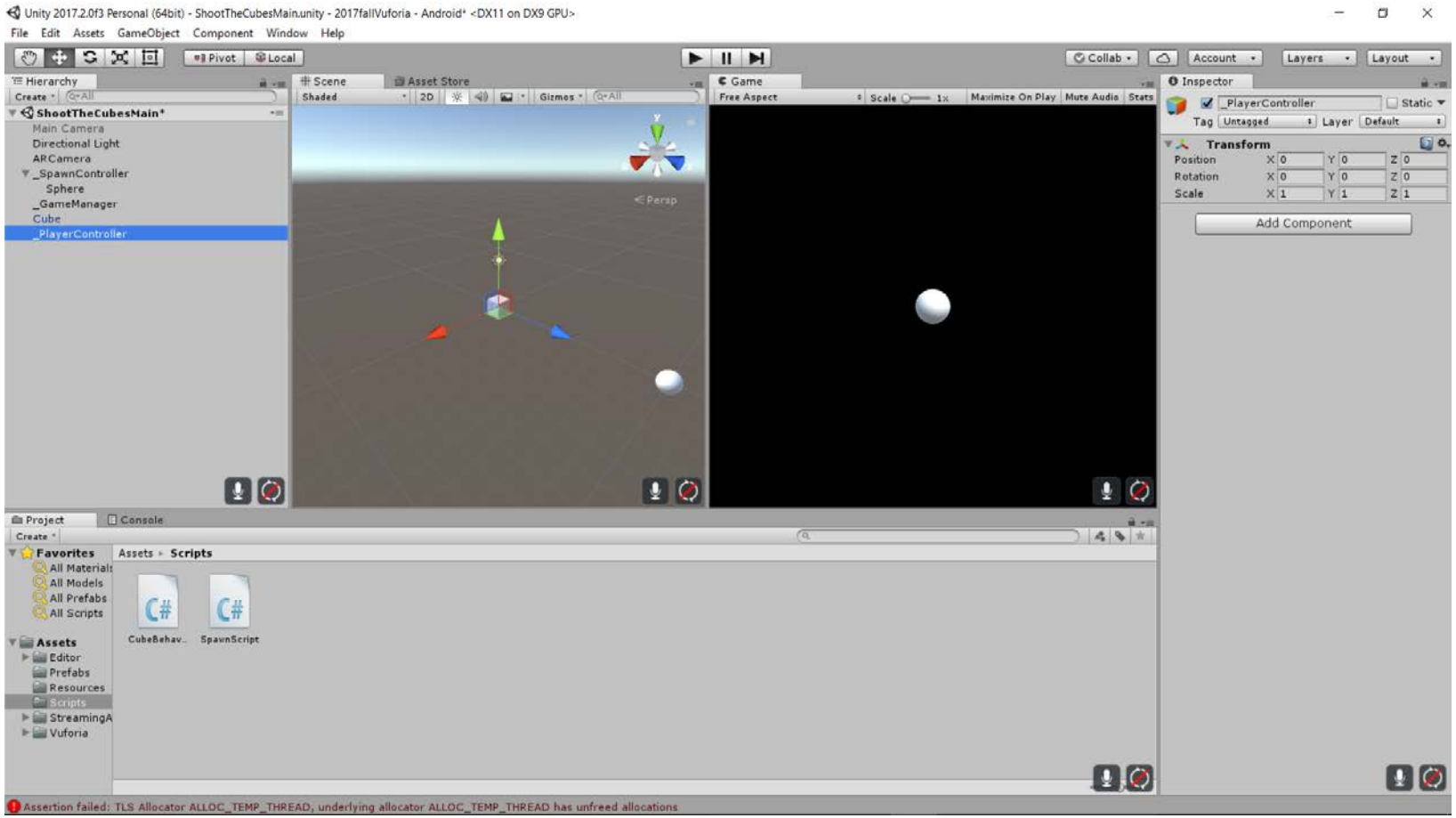
```
59      {
60          // makes the cube orbit and rotate
61          RotateCube();
62
63          // scale cube if needed
64          if (!mIsCubeScaled)
65              ScaleObj();
66      }
67
68      // Makes the cube rotate around a anchor point and rotate around its own axis
69      private void RotateCube()
70      {
71          // rotate cube around camera
72          transform.RotateAround(
73              mOrbitAnchor.position, mOrbitDirection, mOrbitSpeed * Time.deltaTime);
74
75          // rotating around its axis
76          transform.Rotate(mOrbitDirection * 30 * Time.deltaTime);
77      }
78
79      // Scale object from 0 to 1
80      private void ScaleObj()
81      {
82
83          // growing obj
84          if (transform.localScale != mCubeMaxScale)
85              transform.localScale = Vector3.Lerp(transform.localScale, mCubeMaxScale, Time.deltaTime * mGrowingSpeed);
86          else
87              mIsCubeScaled = true;
88      }
89  }
```

Der spawner nu tilfældigt roterende firkanter

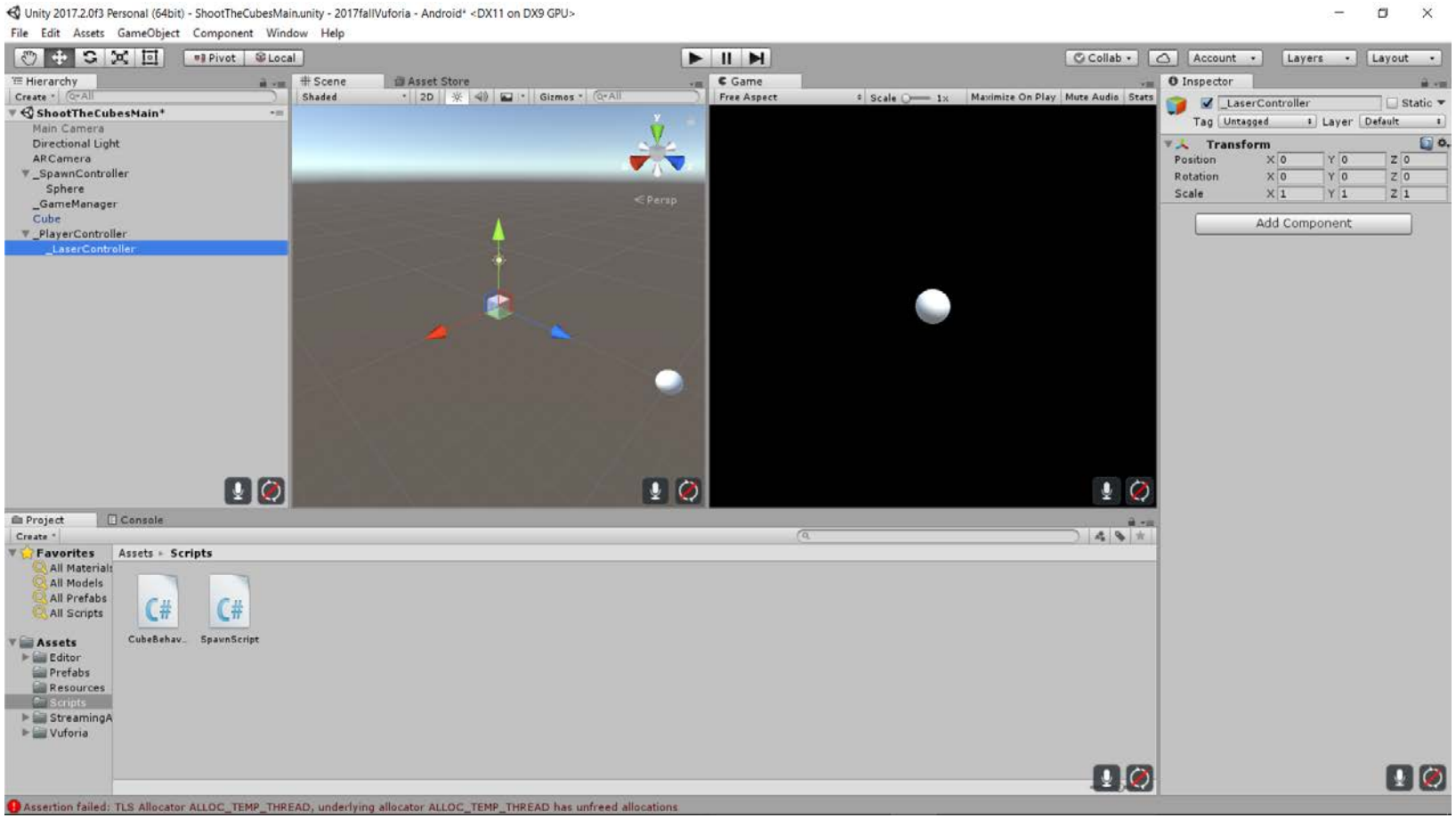


- Der er for mange firkanter! Vi laver en laser der kan skyde dem væk.
- Laseren skal være forbundet til rotationen af ARCamera.
- Hver gang brugeren ”tapper” på skærmen skal der skydes en laser.
- Physics.Raycast bruges til at tjekke om laseren har ramt målet og i så fald fjerne health fra den.

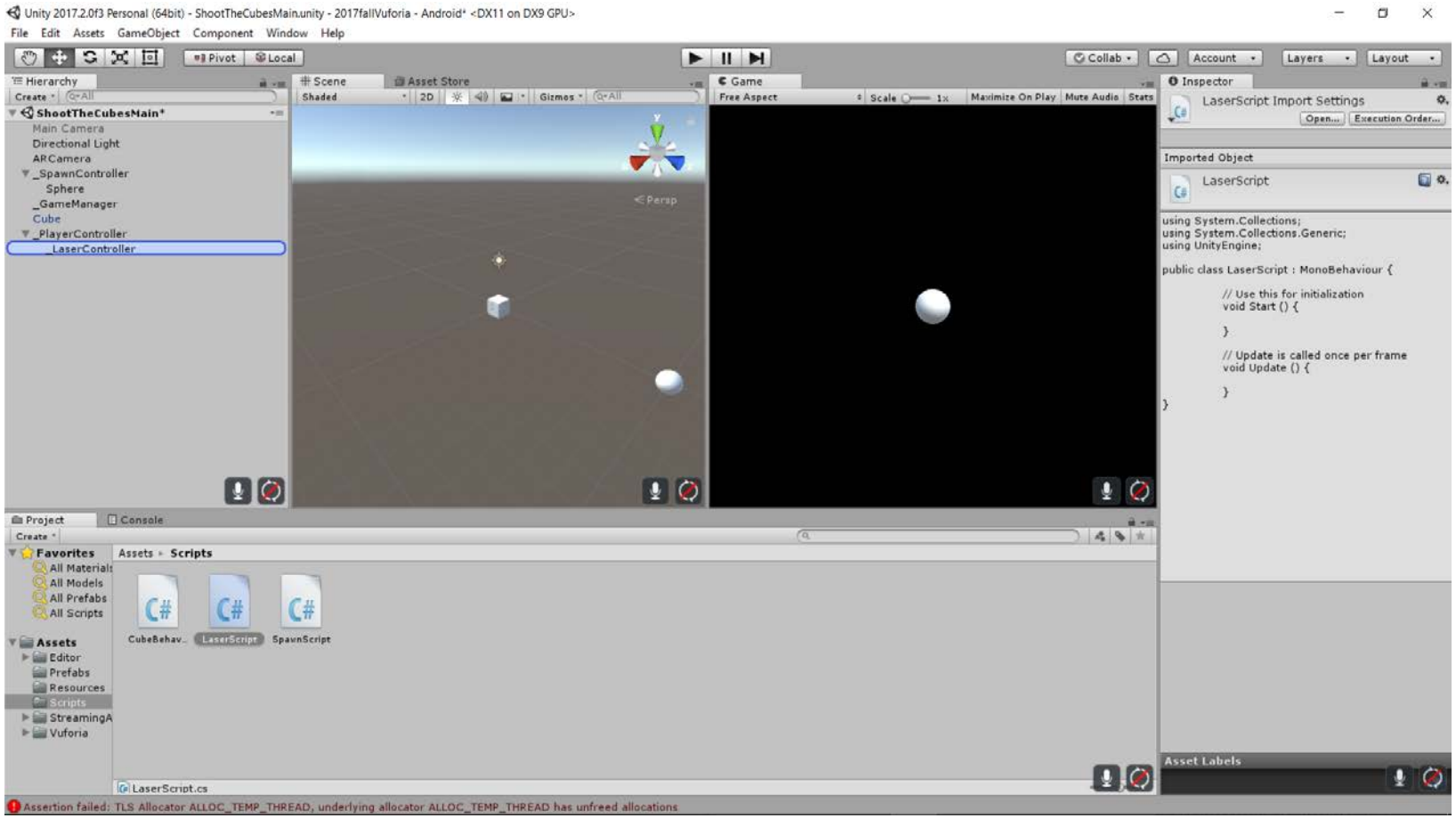
Lav et Empty gameobject kaldet _PlayerController



Lav et nyt empty object under den kaldet _LaserController



Vi laver et script der hedder LaserScript og føjer det til _LaserController



- I LaserScript bruger vi en LineRenderer til at vise laserstrålen ved hjælp af et oprindelsespunkt, der er forbundet til bunden af ARCamera.
- For at få laserstråles oprindelsespunkt - den virtuelle pistols tromle - tager vi kameraets Transform i det øjeblik, hvor en laser er skudt og flytter den 10 enheder ned.
- Vi begynder med at definere variabler til at kontrollere laser indstillingerne og få mLaserLine.

Rediger LaserScript

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class LaserScript : MonoBehaviour
5   {
6
7       public float mFireRate = .5f;
8       public float mFireRange = 50f;
9       public float mHitForce = 100f;
10      public int mLaserDamage = 100;
11
12      // Line render that will represent the Laser
13      private LineRenderer mLaserLine;
14
15      // Define if laser line is showing
16      private bool mLaserLineEnabled;
17
18      // Time that the Laser lines shows on screen
19      private WaitForSeconds mLaserDuration = new WaitForSeconds(0.05f);
20
21      // time of the until the next fire
22      private float mNextFire;
23
24      // Use this for initialization
25      void Start()
26      {
27          // getting the Line Renderer
28          mLaserLine = GetComponent<LineRenderer>();
29      }
30  }
```

- Funktionen der star for at skyde er Fire().
  - Den bliver kaldt hver gang spilleren trykker skyd knappen.
- Camera.main.transform bruges til at få ARCamera position og rotation og laseren placers 10 enheder under disse.
  - Dette placerer laseren i bunden af kameraet.

Rediger LaserScript – endelig udgave

```
1   using UnityEngine;
2   using System.Collections;
3
4   public class LaserScript : MonoBehaviour
5   {
6
7       public float mFireRate = .5f;
8       public float mFireRange = 50f;
9       public float mHitForce = 100f;
10      public int mLaserDamage = 100;
11
12      // Line render that will represent the Laser
13      private LineRenderer mLaserLine;
14
15      // Define if laser line is showing
16      private bool mLaserLineEnabled;
17
18      // Time that the Laser lines shows on screen
19      private WaitForSeconds mLaserDuration = new WaitForSeconds(0.05f);
20
21      // time of the until the next fire
22      private float mNextFire;
23
24      // Use this for initialization
25      void Start()
```

Rediger LaserScript – endelig udgave

```
25      void Start()
26      {
27          // getting the Line Renderer
28          mLaserLine = GetComponent<LineRenderer>();
29      }
30
31      // Update is called once per frame
32      void Update()
33      {
34          if (Input.GetButton("Fire1") && Time.time > mNextFire)
35          {
36              Fire();
37          }
38      }
39
40      // Shot the Laser
41      private void Fire()
42      {
43          // Get ARCamera Transform
44          Transform cam = Camera.main.transform;
45
46          // Define the time of the next fire
47          mNextFire = Time.time + mFireRate;
48
49          // Set the origin of the RayCast
50          Vector3 rayOrigin = cam.position;
51
```

Rediger LaserScript – endelig udgave

```
52          // Set the origin position of the Laser Line. It will always 10 units down from the ARCamera
53          mLaserLine.SetPosition(0, transform.up * -10f);
54
55          // Hold the Hit information
56          RaycastHit hit;
57
58          // Checks if the RayCast hit something
59          if (Physics.Raycast(rayOrigin, cam.forward, out hit, mFireRange))
60          {
61
62              // Set the end of the Laser Line to the object hit
63              mLaserLine.SetPosition(1, hit.point);
64
65          }
66          else
67          {
68              // Set the enfo of the laser line to be forward the camera
69              // using the Laser range
70              mLaserLine.SetPosition(1, cam.forward * mFireRange);
71          }
72
73          // Show the Laser using a Coroutine
74          StartCoroutine(LaserFx());
75      }
76
```

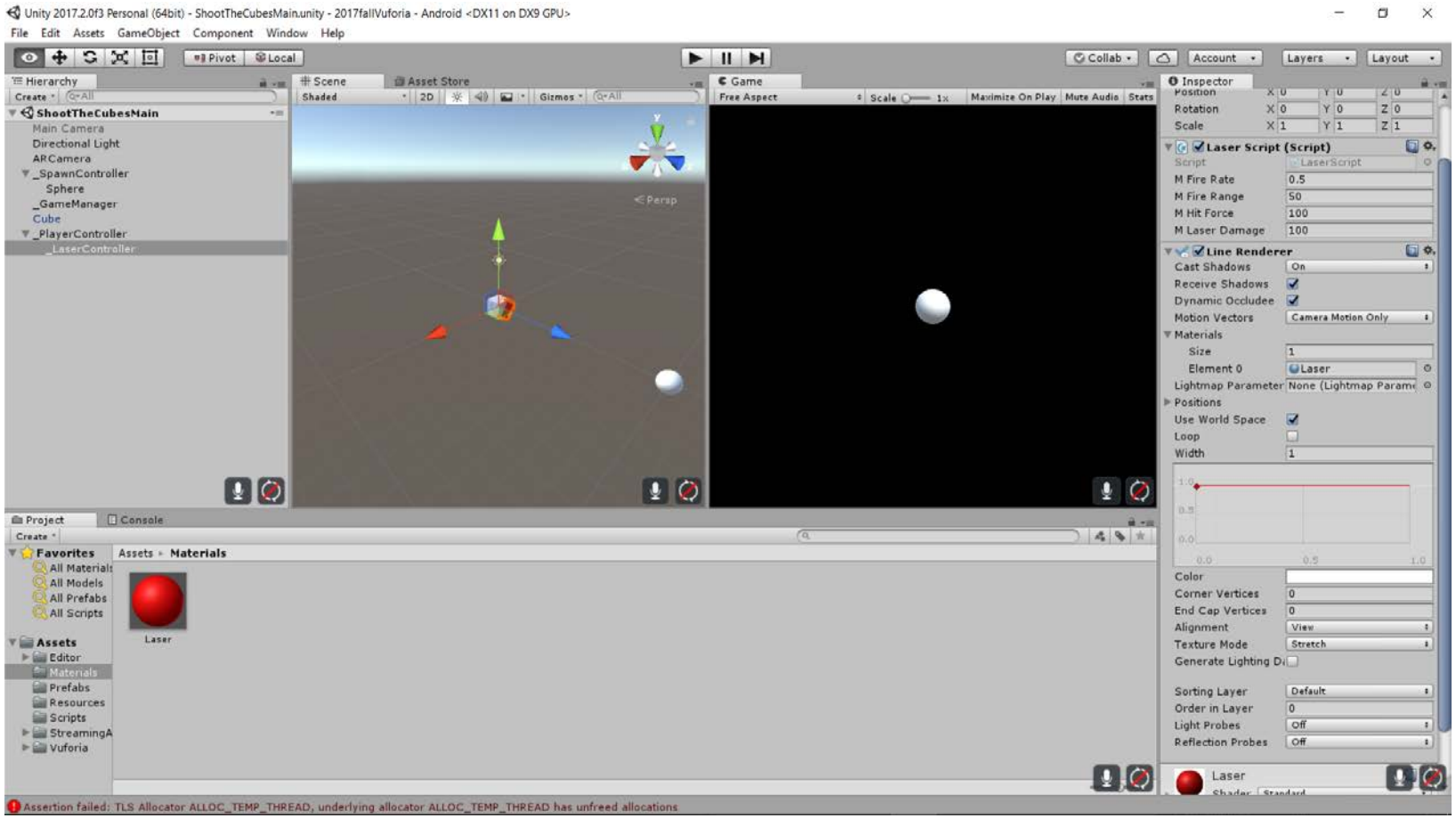
Rediger LaserScript – endelig udgave

```
73              // Show the Laser using a Coroutine
74              StartCoroutine(LaserFx());
75          }
76
77          // Show the Laser Effects
78          private IEnumerator LaserFx()
79          {
80              mLaserLine.enabled = true;
81
82              // Way for a specific time to remove the LineRenderer
83              yield return mLaserDuration;
84              mLaserLine.enabled = false;
85          }
86      }
```

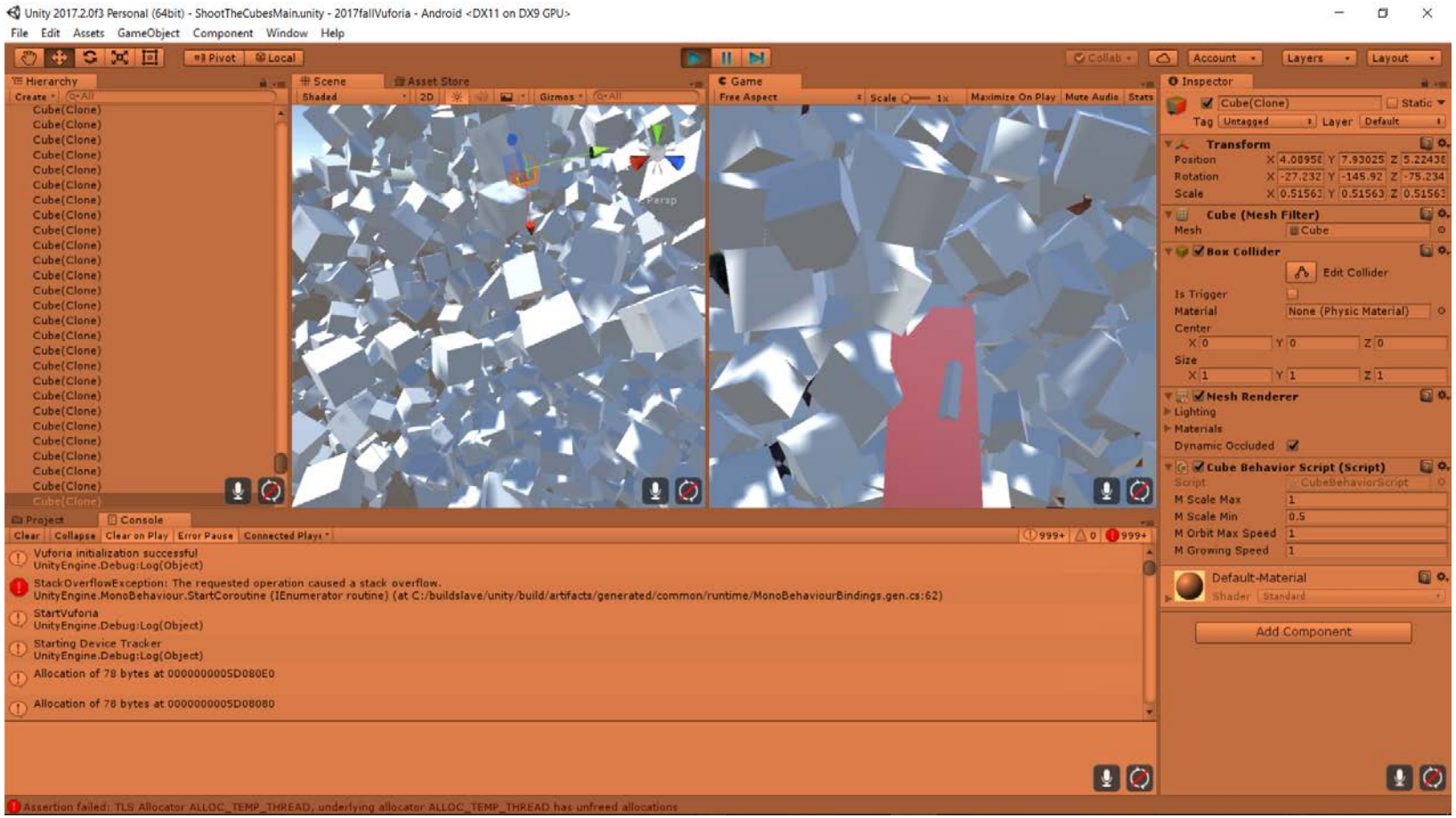
I Unity føjer vi en LineRenderer component til _LaserController object

- Vælg _LaserController og i Inspector vinduet klik på Add Component. Kald den "Line Renderer" og vælg den ny component.
- Lav en ny mappe kaldet Materials, og lav et nyt materiale kaldet Laser.
- Vælg Laser materialet ændr det til en farve efter eget valg.
- Vælg _LaserController og træk Laser materialet til Materials feltet af LineRenderer component.
- Stadig i LineRenderer, under Parameters sæt Start With til 1 og End With til 0.

I Unity føjer vi en LineRenderer component til _LaserController object



Vi skyder nu en laser

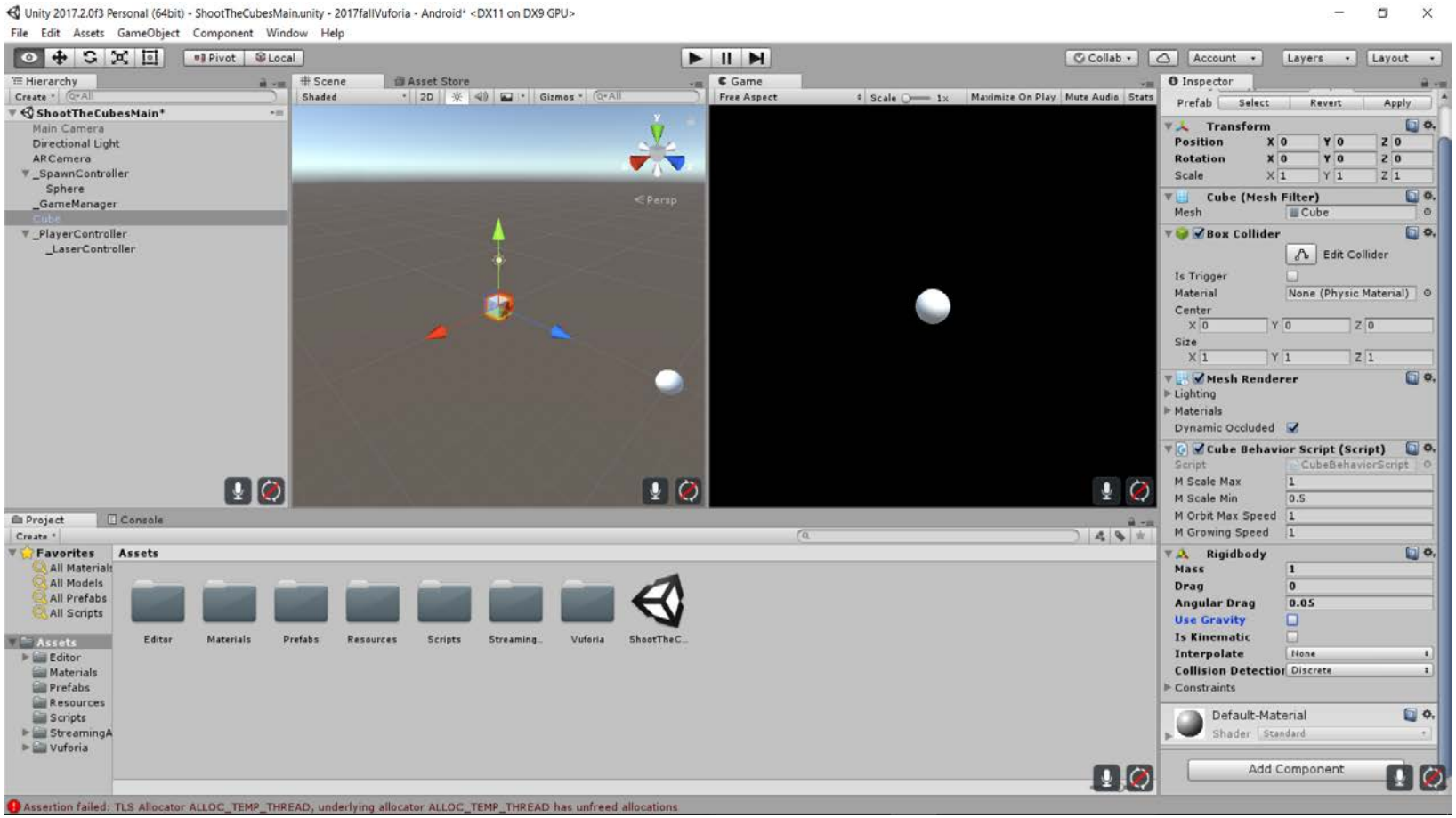


Vores lasere skal ødelægge kuberne

Vi skal ramme målene (kuberne), gøre skade og eventuelt ødelægge kuberne.
Træk vores kube prefab fra prefabs mappen til et hvilket som helst sted på stage.

For at gøre ødelægge dem er vi nødt til at føje en RigidBody til kuberne

På RigidBody komponenten sætter vi Gravity og Is Kinematic til Off



CubeBehavior scriptet redigeres så der kommer en function der kan tilføje skade til kuberne og en anden der vil ødelægge dem når deres health går under 0

```
87              mIsCubeScaled = true;
88          }
89
90          // Cube Health
91          public int mCubeHealth = 100;
92
93          // Define if the Cube is Alive
94          private bool mIsAlive = true;
95
96          // Cube got Hit
97          // return 'false' when cube was destroyed
98          public bool Hit(int hitDamage)
99          {
100             mCubeHealth -= hitDamage;
101             if (mCubeHealth >= 0 && mIsAlive)
102             {
103                 StartCoroutine(DestroyCube());
104                 return true;
105             }
106             return false;
107         }
```

Rediger CubeBehavior

```
107     }
108
109     // Destroy Cube
110     private IEnumerator DestroyCube()
111     {
112         mIsAlive = false;
113
114         // Make the cube desappear
115         GetComponent<Renderer>().enabled = false;
116
117         // We'll wait some time before destroying the element.
118         // This is usefull when using some kind of effect like a explosion sound effect.
119         // In that case we could use the sound lenght as waiting time
120         yield return new WaitForSeconds(0.5f);
121         Destroy(gameObject);
122     }
123 }
```

Kuberne tager nu skade og ødelægges.

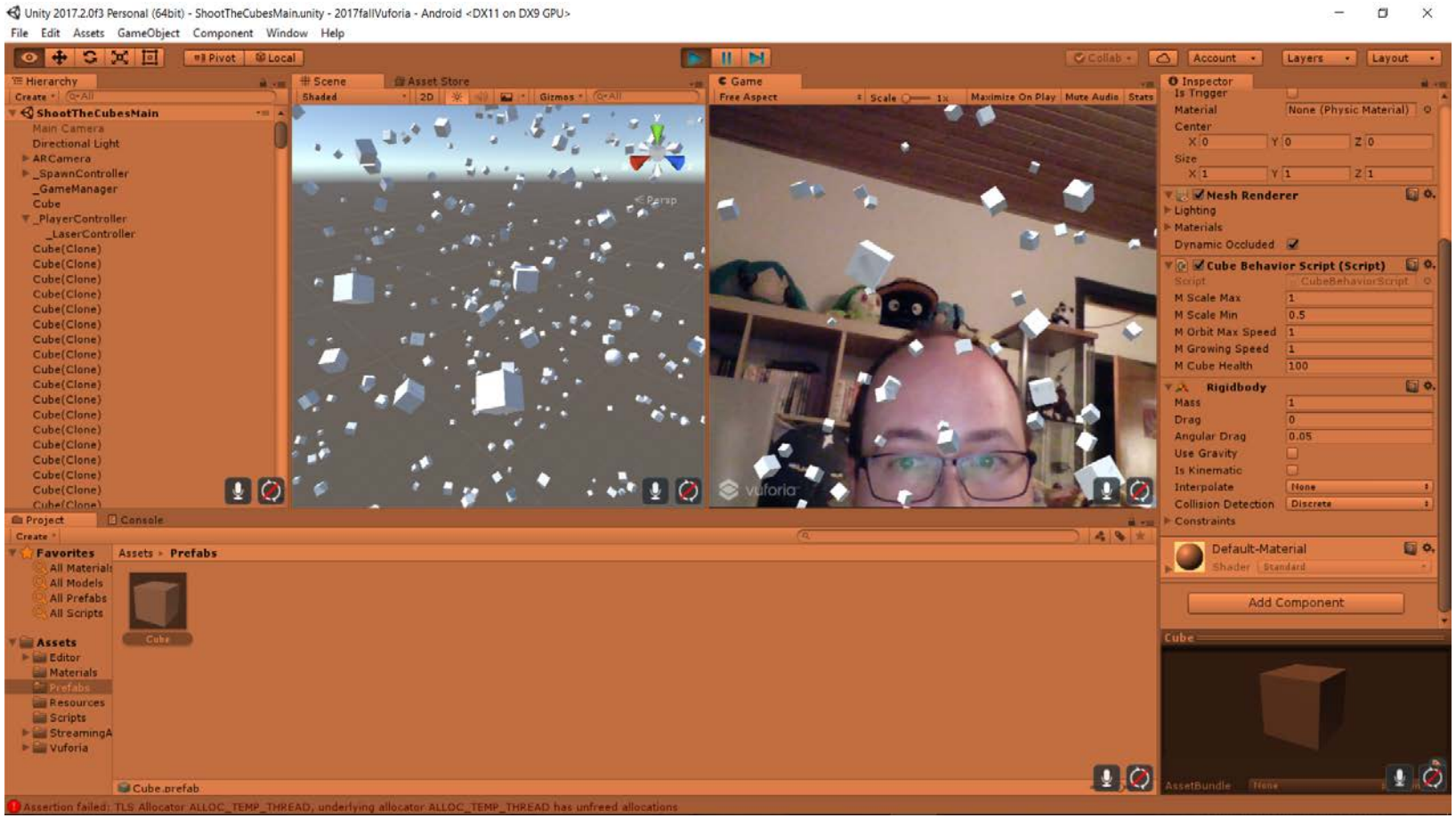
Som det næste redigerer vi LaserScript for at føje skade til kuben.

Vi skal blot ændre Fire() function til at kilde Hit method i CubeBehavior scriptet.

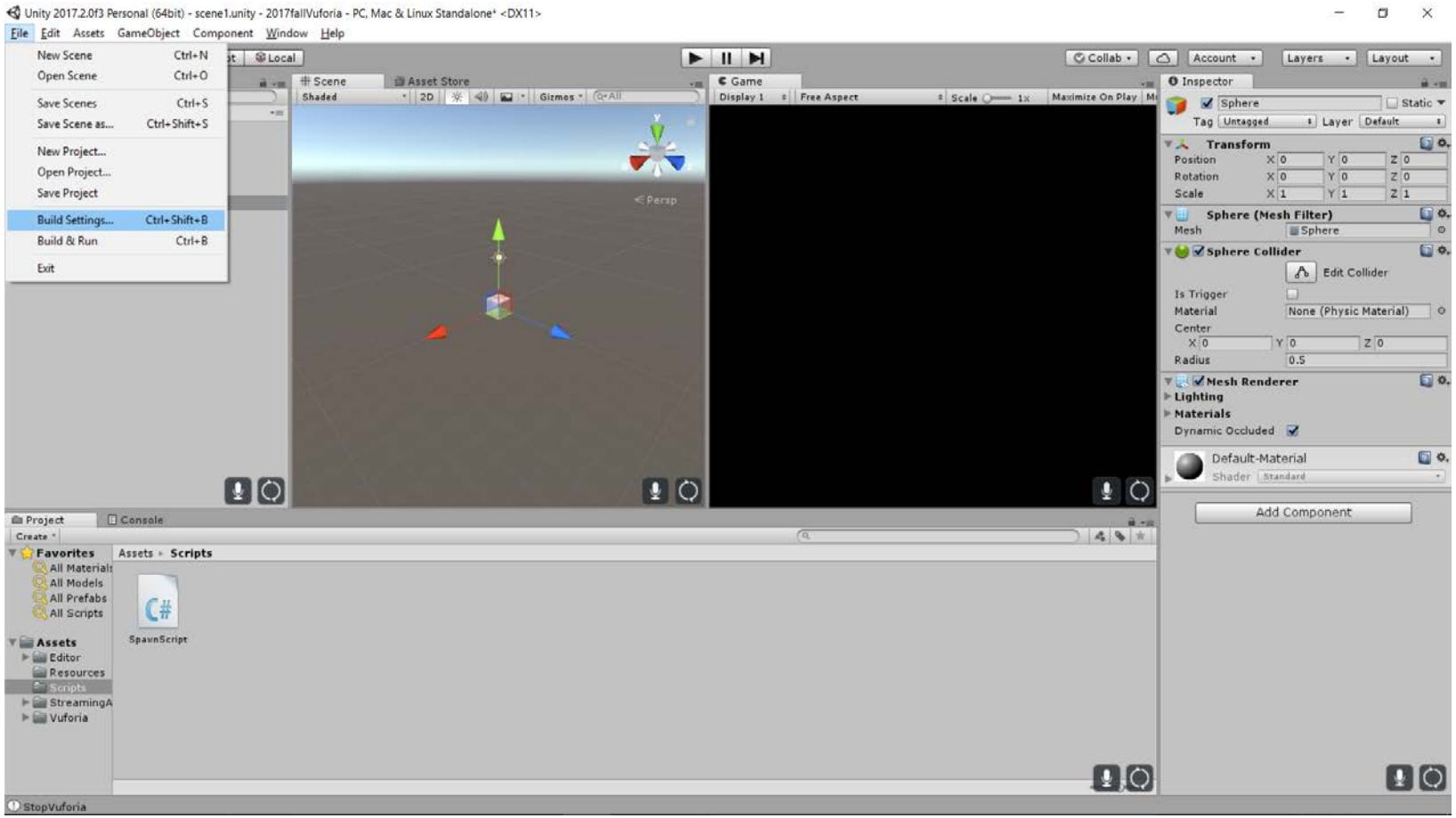
Rediger LaserScript - erstat linje 59 til 65

```
58          // Checks if the RayCast hit something
59          if (Physics.Raycast(rayOrigin, cam.forward, out hit, mFireRange))
60          {
61
62              // Set the end of the Laser Line to the object hitted
63              mLaserLine.SetPosition(1, hit.point);
64
65              // Get the CubeBehavior script to apply damage to target
66              CubeBehaviorScript cubeCtr = hit.collider.GetComponent<CubeBehaviorScript>();
67              if (cubeCtr != null)
68              {
69                  if (hit.rigidbody != null)
70                  {
71                      // apply force to the target
72                      hit.rigidbody.AddForce(-hit.normal * mHitForce);
73                      // apply damage the target
74                      cubeCtr.Hit(mLaserDamage);
75                  }
76              }
77          }
```

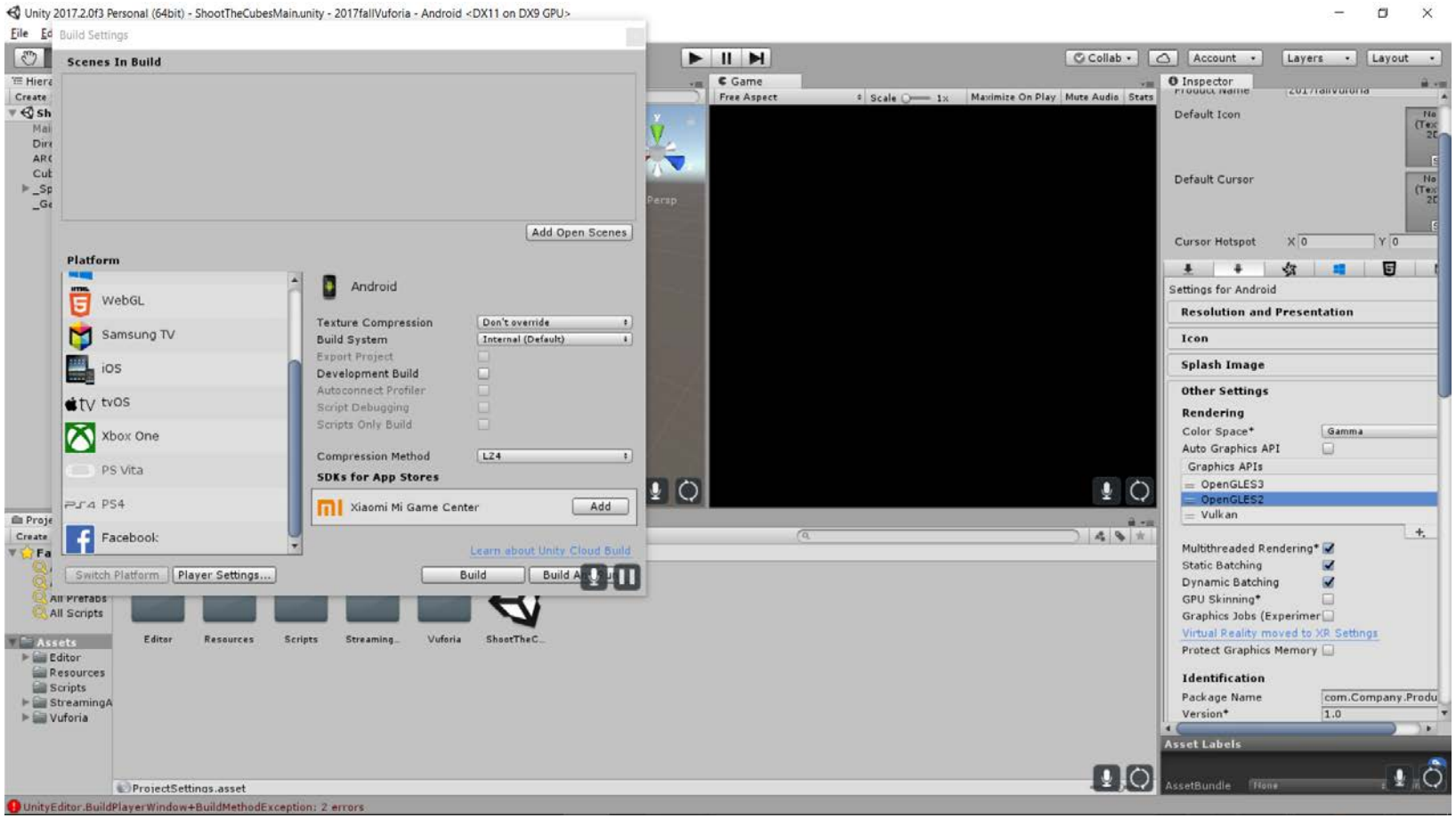
Vores spil burde virke nu



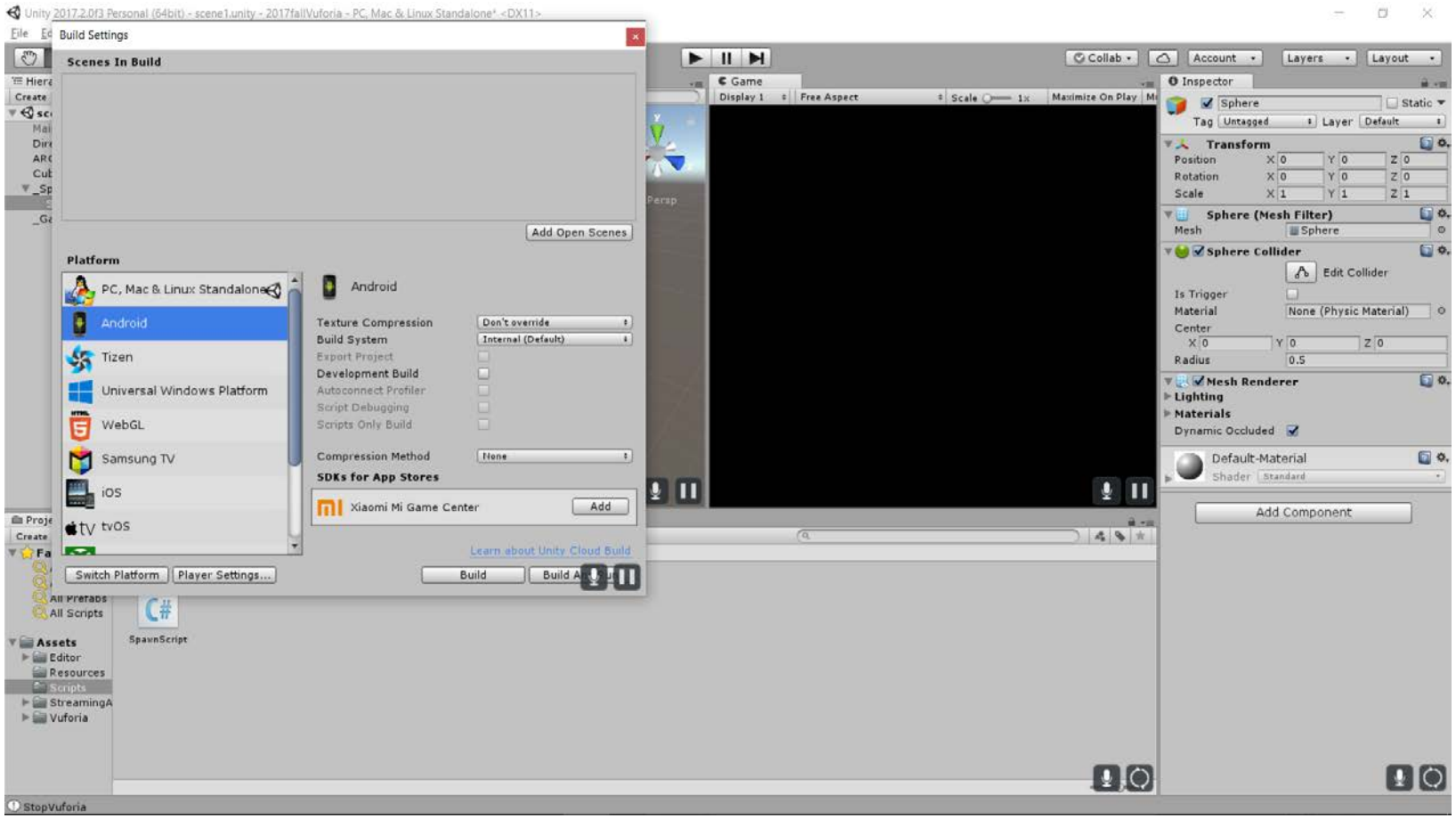
Lad os få den på mobilen.
Åbn Build Settings…



Under Player Settings… slå Auto Graphics API fra og vælg OpemGLES2



Vælg Android og hvis du har Android SDK inde kan du trykke Build (eller installer det)



https://developer.android.com/studio/index.html
https://docs.unity3d.com/Manual/android-sdksetup.html

Tid til eksamensopgave

Kilder

Google Carboard fra Unity

- https://youtu.be/DHBgundyLMU

C# Exception handling

- https://www.tutorialspoint.com/csharp/csharp_exception_handling.htm
- https://youtu.be/gOtZSaLPu-E
- https://youtu.be/El8rlaE3LI8

C# Validation

- https://mva.microsoft.com/en-US/training-courses/programming-in-c-jump-start-14254?l=KkOpp1SfB_8200115888

Kilder

Gratis 3D modeller

- https://www.thepixellab.net/7-sites-for-free-3d-models
- https://www.hongkiat.com/blog/60-excellent-free-3d-model-websites/
- https://free3d.com/3d-models/fbx
- http://www.creativebloq.com/3d/free-3d-models-10121127

Unite Austin 2017

- https://youtu.be/yIvQSrPEtIY

Augmented Reality ved hjælp af Vuforia

- https://code.tutsplus.com/tutorials/creating-ar-games-on-unity-using-vuforia-part-1--cms-27210
- https://code.tutsplus.com/tutorials/pokemon-go-style-augmented-reality-with-vuforia-part-2--cms-27232