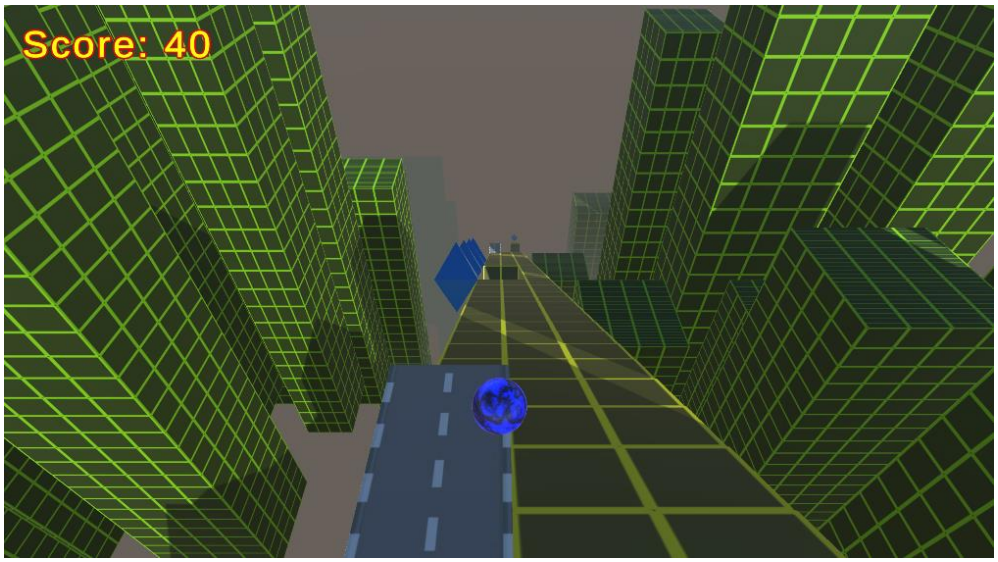
Unity 3D ball endless Guide

Af: Michael Hansen, Coding Pirates Furesø, 2023, version 0.96

Dokumentet ligger her: https://github.com/mhfalken/unity/



Dette er en guide i hvordan man laver et 3D ball endless game, som vist på billedet. Det forudsætter noget kendskab til Unity, og at man selv kan skrive lidt C# kode og finde rundt i GUI'en.
Banen bliver bygget op ved hjælp af kode, så selve bane designet at noget anderledes lavet end ved de andre guides, hvor man bygger banen i GUI'en. Her er heller ikke nogen start pakke, så den smule grafik man skal bruge, må man enten selv lave eller finde på nettet.
Dokumentet og koden er lavet i Unity version **2022.3**. Guiden burde også virker i andre versioner af Unity, men der kan være små forskelle.

1 Simpel bane

1. Start med at oprette et **3D Core** project i Unity og kald det noget fornuftigt.

Vi skal nu lave et 3D ball spil, som bliver ved i det uendelige – dvs. at der skal blive ved med at komme mere bane. Vi kan derfor ikke lave banen i Unity GUI, da den opgave ville blive uendelig stor 😉. Vi bliver derfor nød til at generere banen af noget kode efterhånden som man kommer frem i spillet. På den anden side så bliver man også nød til at fjerne det stykke af banen som allerede er 'brugt' og som er ude af billedet, da der ellers kommer alt for mange objekter til at Unity og computeren kan håndtere det.
Så vi skal lave et spil, hvor banen bliver genereret af noget kode og fjernet igen af noget andet kode. Det kan lyde indviklet, men det er en helt standard måde at lave den slags spil på.
Denne guide beskriver en metode, hvor man bygger én 'linje' af spillet ad gangen og på den måde bygger banen med alle de ting som skal være der, én linje ad gangen.
Spillets grundelement er en firkantet klods (Cube) som der kan være 3 af ved siden af hinanden.

Vi starter med at lave en klods.

2. I **Hierarchy** højreklik og vælg **3D Object->Cube**. I **Inspector**'en check at **Position** er (0, 0, 0) og **Scale** er (1, 1, 1).
3. Lav en ny folder under **Assets,** og kald den *Prefabs.*
4. Træk vores *Cube* ned i **Prefabs**.
5. Slet Cube fra **Hierarchy**'et.

Vi skal nu lave noget kode, som kan tegne vores bane.

6. Lav en ny folder under **Assets**, og kald den *Scripts*.
7. I **Scripts** folderen, lav et **C# Script** og kald det *RoadSpawner*.
8. Åben **RoadSpawner** og tilføj følgende linjer (dem med **fed**).

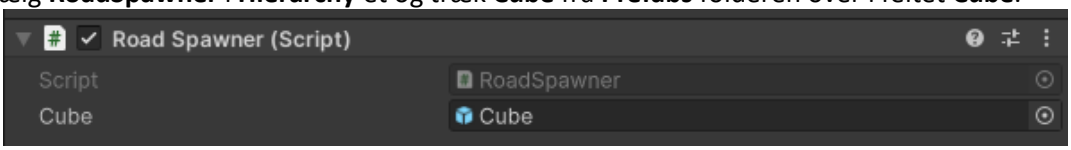
```
[SerializeField] GameObject cube;
float roadCurRotationX =35;
Vector3 roadCurPos = Vector3.zero;

// Start is called before the first frame update
void Start()
{
    for (int i = 0; i < 10; i++)
        RoadAddBlock();
}

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

}

void RoadAddBlock()
{
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, 0);

    Instantiate(cube, roadCurPos, rot);
    roadCurPos += rot * Vector3.forward + 0.02f * Vector3.down;
}
```

(Sidste linje ser lidt mærkelig ud, men det skyldes en uhensigtsmæssighed i Unity, som er forsøgt fikset.)

Lige nu sker der ingenting, da vores script ikke er tilknytte noget i Unity.

9. Lav et tomt objekt (**Create Empty)** og kald det *RoadSpawner*.

10. Træk vores **RoadSpawner** script op over **RoadSpawner** i **Hierarchy**'et.
11. Vælg **RoadSpawner** i **Hierarchy**'et og træk **Cube** fra **Prefabs** folderen over i feltet **Cube**.



Kør spillet og se at der kommer 10 kasser frem.
(Kameraet står sikkert lidt skidt, men det løser vi senere, når vi sætter vores kugle ind.)
Vores bane er lidt smal, så vi skal have flere klodser ved siden af hinanden.

12. Tilføj følgende linjer i `RoadAddBlock` (dem med **fed**):

```
Instantiate(cube, roadCurPos + rot * Vector3.left, rot);
Instantiate(cube, roadCurPos, rot);
Instantiate(cube, roadCurPos + rot * Vector3.right, rot);
```

(Linjerne er forberedt på at vi senere laver en hældning på banen.)
Kør spillet og se at banen er blevet bredere.
Lige nu er banen meget simpel, men vi laver den mere avanceret senere.

2 Kuglen

Vi skal nu have en kugle til at køre på banen og have kameraet til at følge kuglen.

1. I **Hierarchy**'et lav en kugle (**3D Object->Sphere**) og kald den *Player*.
2. Ret **Position.Y** til 1
3. Ret **Scale** til (0.4, 0.4, 0.4)
4. Tilføj **RigidBody** og ret følgende felter:

Kør spillet og se at kuglen kører ned ad banen.
Vi skal nu have kameraet til at følge efter kuglen, så man kan se hvad der sker.

5. I **Scripts** folderen opret et nyt script og kald det *CameraFollower*.
6. Åben **CameraFollower** scriptet og ret koden så den ser sådan ud:

```
public Transform player;
public Vector3 offset;

void Update()
{
    transform.position = player.position + offset;
}
```

7. Træk **CameraFollower** scriptet op over **Main Camera** i **Hierarchy**'et.
8. Vælg **Main Camera** i **Hierarchy**'et og ret **Rotation.X** til *42*.
9. Træk **Player** over i **Player** feltet i **Inspector**'en.
10. I **Inspector->Camera Folower** ret også **Offset** til (0, 3, -2).

Kør spillet og se at kameraet følger efter kuglen ned ad banen.

2.1 Kuglekontrol

Lige nu kører kuglen bare ned ad banen og kan ikke styres. Det skal vi nu have gjort noget ved.

1. I **Scripts** folderen lav et nyt script og kald det *PlayerController*.
2. Træk scriptet op over **Player** i **Hierarchy**'et.
3. Ret scriptet, så koden ser sådan ud:

```
Rigidbody rb;
bool lastMove;

// Start is called before the first frame update
void Start()
{
    rb = GetComponent<Rigidbody>();
}

// Update is called once per frame
void Update()
{
    float moveX = Input.GetAxis("Horizontal");

    if ((moveX != 0) || lastMove)
        rb.velocity = new Vector3(3 * moveX, rb.velocity.y, rb.velocity.z);
    if (moveX != 0)
        lastMove = true;
    else
        lastMove = false;
}
```

(Koden ser lidt 'mærkeligt' ud, men det skyldes at kuglen på sigt også bliver påvirket af at banen hælder til siderne, og det kræver at koden er lidt mere avanceret.)

Kør spillet og se at man kan styre kuglen fra side til side.

3 Grafik

Spillet ser lidt kedeligt ud med hvide klodser og en hvid kugle. Det skal der nu rettes op på.

3.1 Klodsgrafik

For klodserne er løsningen at sætte et billede på siden af klodsen, så man kan se noget struktur og dybde i billedet.

1. Start med at lave en folder, som hedder *Graphics*.
2. Find eller lav et billedet som er kvadratisk og har en tydelig kant som på billedet og læg det i folderen *Graphics*. (Træk det over i folderen.) Det må meget gerne se anderledes ud, men det er lidt vigtigt for spillet at der er nogle tydelige linjer, for at få den rigtige effekt.
   Alternativt kan man tage et screenshot af billedet oven over og så bruge det.
3. Åben **Cube Prefab**'en og træk nu billedet op over **Cube** i **Hierarchy**'et. Unity laver automatisk det materiale som skal bruges.

Kør spillet og se forskellen. Man kan trimme lidt på farverne ved at åbne det materiale som Unity har lavet. Det ligger under *Graphics/Materials*.

3.2 Kuglegrafik

Kuglen skal også have noget grafik, men det er lidt sværere at lave selv, så her er det nemmere at finde noget på nettet. Et godt sted at kikke er i Unity asset store: https://assetstore.unity.com/
I mit spil har jeg brugt: *34 Free Shader FX Masks*.
Når man har fundet og installeret den grafik man vil bruge, så gør man følgende:

1. Vælg det materiale som man ønsker at bruge og træk det op over **Player** i **Hierarchy**'et.

Hvis man gerne vil ændre farven, så gør følgende:

2. Vælg **Player** i **Hierarchy**'et og i **Inspector**'en under **Matrial** ret **Color** til den farve man ønsker.

4 Baneelementer

Nu begynder det at ligne noget, men banen er lidt ensformig og kort.

1. Gør banen lidt længere, ved at ændre i `for` løkken i `Start()`. Ændre fx 10 til 15.

Det ændrer dog ikke på at banen er ensformig.

4.1 Klodsmaske

Vi skal derfor have ændret i `RoadAddBlock()` funktionen, så den kan lave forskellige elementer. Man kunne også i stedet for lave en masse forskellige funktioner, én for hvert baneelement type, men det er sværere at vedligeholde og vi har brug for at ændre i funktionen flere gange i løbet af denne guide.
Det første vi skal have vores `RoadAddBlock()` funktion til er at kunne, er at vælge hvilke(n) klods(er) man vil have – se billede.

Det gør vi ved at tilføje en parameter som er en maske, så man kan vælge hvilke klodser som man vil have, dvs. man kan fx lave en bane med et hul i midten som på billedet oven over.

2. Ændre nu koden så den ser sådan ud (ændringer med **fed**):

```
[SerializeField] GameObject cube;
float roadCurRotationX =35;
Vector3 roadCurPos = Vector3.zero;

const int RB_L = 0x4;  // Left
const int RB_C = 0x2;  // Center
const int RB_R = 0x1;  // Right
const int RB_A = RB_L + RB_C + RB_R;  // All

void Start()
{
    for (int i = 0; i < 10; i++)
        RoadAddBlock(RB_A);
}

void Update()
{

}

void RoadAddBlock(int mask)
{
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, 0);

    if ((RB_L & mask) > 0)
    {
        Instantiate(cube, roadCurPos + rot * Vector3.left, rot);
    }
    if ((RB_C & mask) > 0)
    {
        Instantiate(cube, roadCurPos, rot);
    }
    if ((RB_R & mask) > 0)
    {
        Instantiate(cube, roadCurPos + rot * Vector3.right, rot);
    }

    roadCurPos += rot * Vector3.forward + 0.02f * Vector3.down;
}
```

Kør spillet og se at alt stadig virker som før.
Nu kan man begynde at lave en bane, ved at tilføje flere `for` løkker i `Start()`.

3. Tilføj følgende linjer i `Start()` (nye linjer er med **fed**):

```
for (int i = 0; i < 10; i++)
    RoadAddBlock(RB_A);
for (int i = 0; i < 5; i++)
    RoadAddBlock(RB_L);
```

Kør spillet og se at der er kommet lidt mere bane, hvor kun den venstre kasse er med og stykket er kortere, kun 5 klodser.
Man kan sætte `RB_L` (left), `RB_C` (center) og `RB_R` (right) sammen med `+` og på den måde vælge flere klodser på en gang. Man laver en split bane ved at bruge `RB_L+RB_R`.

4. Tilføj nu nogle flere `for` løkker med forskellige længde og klodser.

4.2 Klodshældning til siden

Nu skal vi gøre så klodserne kan hælde til en af siderne (se billedet længere nede). Her tilføjer vi endnu en parameter til `RoadAddBlock()`, men på en sådan måde at man kun behøver at angive den, hvis den er forskellig fra 0.

1. Ret følgende linjer (ændringer med **fed**):

```
void RoadAddBlock(int mask, float angleZ = 0)
{
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, angleZ);
```

Kør programmet og se at alt stadig virker.
Tilføj følgende `for` løkke efter de andre `for` løkker i `Start()`:

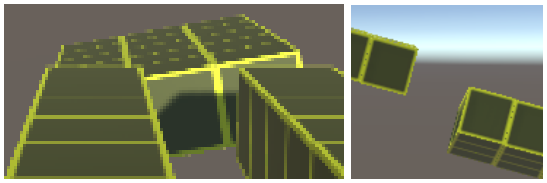
```
for (int i = 0; i < 5; i++)
    RoadAddBlock(RB_A, 10);
```

Kør programmet og se at banen nu hældet til den ene side. Det 'nye' tal angiver hældningen i grader og fortegnet (plus/minus) angiver, hvilken retning banen hælder.

2. Tilføj nu nogle flere `for` løkker, som bruger den nye hældning.

4.3 Klodsafstand

Det ser lidt mærkeligt ud når banen ændrer hældning, da der kommer nogle 'bump'. Løsningen er at lave lidt afstand og et trin ned. Se billeder.



1. Tilføj følgende funktion til **RoadSpawner** scriptet:

```
void RoadAddSpace(float y, float z)
{
    roadCurPos += new Vector3(0, y, z);
}
```

Funktionen bruges på følgende måde:

```
RoadAddSpace(-1, 1);
```

Det første tal er op/ned og det andet er afstanden til næste element. Denne linje laver en 'afstand' på banen, som går én kasse ned og springer én kasse over. Se billederne oven over. Funktionen kan også bruges til at lave en kløft man skal over.

2. Brug funktionen til at få lavet en bedre bane med.

Senere i guiden, så laver vi nogle flere baneelementer som forhindringer og hop, men vi skal lige have ordnet nogle vigtige ting først.

5 Banesegmenter

Problemet med den måde at lave banen på er, at hele banen bliver lavet med det samme og det er jo lidt svært med en uendelig bane og det bliver meget stort! Vi skal derfor have vores bane delt op i segmenter, hvor hvert segment indeholder et stykke af banen, fx 30-50 elementer.

1. Lav en funktion som hedder `RoadSegment1()` og kopier banen (`for` løkkerne) over i den. Koden skal se ca. sådan ud:

```
void RoadSegment1()
{
    for (int i = 0; i < 10; i++)
        RoadAddBlock(RB_A);
    for (int i = 0; i < 5; i++)
        RoadAddBlock(RB_R);
    for (int i = 0; i < 5; i++)
        RoadAddBlock(RB_L + RB_R);
    RoadAddSpace(-1, 1);
    for (int i = 0; i < 5; i++)
        RoadAddBlock(RB_A, 10);
}
```

Hvis banen allerede er meget lang, så lav flere funktioner, hvor du ændrer tallet `1` (i RoadSegment1) til `2`, `3` osv.

2. Kald nu `RoadSegment1()` funktionen fra `Start()`.

```
void Start()
{
    RoadSegment1();
    RoadSegment2();  // Hvis der er flere funktioner
}
```

Nu skal du gerne have nogle funktioner, som hedder **RoadSegment** + tal. Husk at kalde alle funktionerne i `Start()`. Det er vigtigt at du har mere end én **RoadSegment** funktion.

Alt er stadig som før, hvor vi laver hele banen med det samme. Det skal nu laves om, så vi kalder `RoadSegment` funktionerne efterhånden som vi gennemfører banen. Til det formål skal vi bruge noget lidt mere avanceret C# kode.

3. Lav en liste af funktionspointere (i **RoadSpawner** filen):

```
List<Action> roadFunctions = new List<Action>();
int roadFuncIndex;
```

4. Tilføj denne linje under de andre `using` i toppen af filen

```
using System;
```

5. Ændrer nu `Start()` så den ser sådan ud:

```
void Start()
{
    roadFunctions.Add(() => RoadSegment1());
    roadFunctions.Add(() => RoadSegment2());
}
```

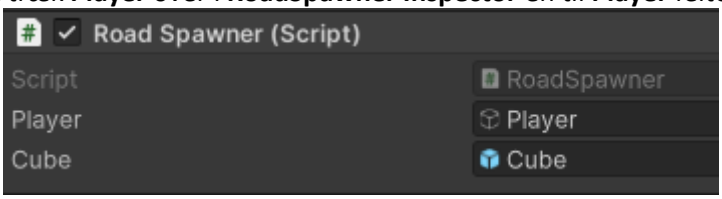
Her skal være én linje for hver **RoadSegment** funktion man har.

`Update()` funktionen skal nu ændres, så den kalder vores **RoadSegment** funktioner efterhånden som vi kommer gennem banen.

6. Ændrer nu `Update()` så den ser sådan ud:

```
void Update()
{
    if (player.transform.position.z > (roadCurPos.z - 30))
    {
        if (roadFuncIndex < roadFunctions.Count)
        {
            roadFunctions[roadFuncIndex]();
            roadFuncIndex++;
        }
    }
}
```

7. Fra **Hierarchy**'et træk **Player** over i **RoadSpawner Inspector**'en til **Player** feltet.



Kør spillet og se hvordan banen bliver lavet efterhånden som man kommer frem.

5.1 Cleanup

Nu får vi bygget vores bane lidt af gangen, men den ender med at blive uendelig stor. Det er derfor vigtigt at man fjerner banen og alle objekter når de ikke længere er synlige, da der ellers kommer alt for mange objekter.
Vi laver en liste af alle de objekter som vi putter ind i spillet og så laver vi en funktion som fjerne alle de objekter som ikke bruges mere.

1. Tilføj en liste til at holde alle elementer i spillet (i **RoadSpawner** scriptet).

```
List<GameObject> objList;
```

2. Opret en tom liste (tilføj linje med **fed**):

```
void Start()
{
    objList = new List<GameObject>();
    roadFunctions.Add(() => RoadSegment1());
```

Sæt alle vores klodser ind i listen efterhånden som de bliver genereret.

3. Ændret `RoadAddBlock()` funktionen, så den ser sådan ud (nye ting med **fed**):

```
void RoadAddBlock(int mask, float angleZ = 0)
{
    GameObject obj;
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, angleZ);

    if ((RB_L & mask) > 0)
    {
        obj = Instantiate(cube, roadCurPos + rot * Vector3.left, rot);
        objList.Add(obj);
    }
    if ((RB_C & mask) > 0)
    {
        obj = Instantiate(cube, roadCurPos, rot);
        objList.Add(obj);
    }
    if ((RB_R & mask) > 0)
    {
        obj = Instantiate(cube, roadCurPos + rot * Vector3.right, rot);
        objList.Add(obj);
    }

    roadCurPos += rot * Vector3.forward + 0.02f * Vector3.down;
}
```

Selve oprydningsfunktion bliver lidt 'indviklet', da den senere også skal bruges til andre objekter som kommer ind i spillet.

4. Tilføj følgende funktionen:

```
void ObjCleanUp()
    {
        GameObject obj;
        // Delete 'old' elements
        for (;;) {
            if (objList.Count == 0)
                break;
            obj = objList[0];
            if (obj == null)
            {
                objList.RemoveAt(0);
                continue;
            }
            if (obj.transform.position.z > (player.transform.position.z - 10))
                break;
            objList.RemoveAt(0);
            Destroy(obj);
        };
    }
```

Tilføj følgende linje i toppen af `Start()` funktionen:

```
objList = new List<GameObject>();
```

Tilføj denne linje til `Update()` funktionen (den med **fed**):

```
if (player.transform.position.z > (roadCurPos.z - 30))
{
    ObjCleanUp();
    if (roadFuncIndex < roadFunctions.Count)
```

Kør spillet og kik i **Scene** view mens spillet kører, for at se hvordan den 'brugte' bane forsvinder.

6 Udfordringer

Nu skal vi have lavet noget mere game play, såsom at detektere når man dør, forhindringer man skal undgå og hop.

6.1 Game over detect

Udfordringen ved at se hvornår man er død er, at **Player** scriptet ikke rigtig kan 'se' banen, da det er noget Unity automatisk håndterer, så selve detekteringen skal laves i **RoadSpawner** scriptet og så kalde en funktion i **Player** scriptet, som så stopper spillet og genstarter det.

1. Tilføj følgende linje i 'toppen' af **PlayerController** scriptet, under de andre `using` linjer.

```
using UnityEngine.SceneManagement;
```

2. Tilføj følgende linjer i bunden af scriptet (inden den sidste `}`).

```
public void Die()
{
    SceneManager.LoadScene(SceneManager.GetActiveScene().name);
}
```

3. I **RoadSpawner** scriptet tilføj følgende linjer i bunden af `Update()` funktionen:

```
if (player.transform.position.y < roadCurPos.y)
{
    GameObject.Find("Player").GetComponent<PlayerController>().Die();
}
```

Kør spillet og se at man 'dør' hvis man kører ud over kanten. Det tager lige lidt tid, da man jo skal kunne se at det er gået galt.

6.2 Forhindringer

Vi skal nu lave nogle kasser, som man ikke må røre. De laves på samme måde som de kasser, som laver vores bane, men skal se anderledes ud, så man kan skelne dem fra banen.

1. Lav en kasse (Cube) som i starten af guiden, men sæt **Box Collider->Is Trigger**.
2. Giv den en texture så den ser godt ud. (Hvis man bruger det samme 'Material' som til banen, så får de samme farve, så det letteste er at kopiere materialet inden, så de kan får hver sin farve.)
3. Tilføj et **Tag** som hedder *Trap*.
4. Kald den *CubeObstacle*.
5. Lav en **Prefab** af den.
6. Slet den fra **Hierarchy**'et.

7. Tilføj følgende linje til **RoadSpawner** scriptet:

```
[SerializeField] GameObject cubeObstacle;
```

8. Tilføj følgende funktion til **RoadSpawner** scriptet:

```
void ObstacleAddBlock(float x, float y, float angleZ = 0)
{
    GameObject obj;
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, angleZ);
    obj = Instantiate(cubeObstacle, roadCurPos + rot * new Vector3(x, y, 0), rot);
    objList.Add(obj);
}
```

9. Brug denne linje til at lave en forhindring med. Det første tal er x placeringen (-1, 0, 1) og andet tal er højden, hvor 1 svare til at den stå på banen. Man kan også angive et 3. tal som angiver hældningen, ligesom for banen.

```
ObstacleAddBlock(0, 1);
```

10. I **Hierarchy**'et vælg **RoadSpawner** og træk **CubeObstacle** (fra **Prefabs**) over i **Cube Obstacle**.

Kør spillet og se den nye forhindring. Der sker dog ikke noget når man 'rammer' den. Det skal vi nu have tilføjet.

11. Tilføj følgende funktion til **PlayerController** scriptet:

```
private void OnTriggerEnter(Collider other)
{
    if (other.gameObject.CompareTag("Trap"))
    {
        Die();
    }
}
```

Kør spillet og se at det stopper så snart man rammer en forhindring.
Det ser lidt 'brutalt' ud og hvis man synes det stopper lidt for 'hurtigt' må man selv lave et lille delay inden spillet stopper.

6.3 Hop

Vi skal nu lave en lille rampe, som kuglen kan bruge til at hoppe over en kløft med.

1. Lav et Plane og sæt følgende i **Inspector**'en:

2. Kald den *Jump*
3. Find et godt billedet og læg på rampen, så den ser godt ud.
4. Lav en **Prefab** af den.
5. Slet den fra **Hierarchy**'et.

6. Tilføj følgende linje til **RoadSpawner** scriptet:

```
    [SerializeField] GameObject jump;
```

7. Tilføj følgende funktion til **RoadSpawner** scriptet:

```
void RoadAddJump(float x, float y, float angleX)
{
    GameObject obj;
    Quaternion rot = Quaternion.Euler(roadCurRotationX + angleX, 0,
0);
    obj = Instantiate(jump, roadCurPos + new Vector3(x, y, 0), rot);
    objList.Add(obj);
}
```

8. Brug denne linje til at lave et hop med. Det første tal er x placeringen (-1, 0, 1), andet tal er højden – typisk 1 og det sidste tal er rampens vinkel – det skal typisk være et negativt tal.

```
RoadAddJump(-1, 1, -30);
```

9. I **Hierarchy**'et vælg **RoadSpawner** og træk **Jump** (fra **Prefabs**) over i **Jump**.

Kør spillet og se at man nu kan få kuglen til at køre op ad rampen.

7 Items

Vi tilføjer nu nogle ting som man kan tage og så score nogle point.

Start med at finde nogle ting man kan tage. Det letteste er at kikke i Unity Asset store https://assetstore.unity.com/, da de kan importeres direkte ind i Unity.
Jeg har brugt følgende: *Simple Gems Ultimate Animated Customizable Pack.*

1. Træk den valgte figur ind i **Scene**n.
2. Ret størrelsen så den passer med spillet.
3. Tilføj en **Mesh Collider** og vælg **Convex->Is Trigger**.
4. Tilføj et **Tag** som hedder *Item*.
5. Lav en **Prefab**
6. Slet den fra **Hierarchy**'et.

7. Tilføj følgende linje i **RoadSpawner** scriptet.

```
    [SerializeField] GameObject itemGem;
```

8. Tilføj følgende funktion til **RoadSpawner** scriptet.

```
void ItemAddGem(float x, float y, float angleZ = 0)
{
    GameObject obj;
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, angleZ);
    Quaternion rot1 = Quaternion.Euler(roadCurRotationX+90, 0, angleZ);
    obj = Instantiate(itemGem, roadCurPos + rot*new Vector3(x, y, 0), rot1);
    objList.Add(obj);
}
```

9. Træk det items som du netop har lavet over i **RoadSpawner Item Gem**.
10. Brug denne linje til at indsætte items med. Parametrene er de samme som for boks forhindringen.

```
ItemAddGem(1, 1, 10);
```

Kør spillet og se at items'ne kommer frem, men de kan ikke tages endnu.

Tilføj følgende linjer til `OnTriggerEnter()` funktionen.

```
if (other.gameObject.CompareTag("Item"))
{
    Destroy(other.gameObject);
}
```

Kør spillet og se at items forsvinder, nå man rører den.

7.1 Point

Vi skal nu have nogle point, når vi tager en ting i spillet og ens point skal stå på skærmen, mens spillet kører.

1. Tilføj følgende linje til **PlayerController** scriptet.

```
int score;
```

2. Tilføj følgende linje til `OnTriggerEnter()` funktionen (den med **fed**).

```
if (other.gameObject.CompareTag("Item"))
{
    score += 10;
    Destroy(other.gameObject);
}
```

Nu har vi fået lavet point, men vi mangler at få dem frem på skærmen.

3. I **Hierarchy**'et indsæt **UI->Text - TextMeshPro**, og vælg **Import TMP Essentials** på den dialog som kommer frem, og kald feltet *Score*.
4. Vælg **Score** og i **Scenen** flyt feltet til toppen af skærmen.
5. I **Inspector**'en formater feltet så det ser godt ud.
6. Tilføj følgende linje til **PlayerController** scriptet.

```
[SerializeField] TMP_Text scoreText;
```

(Editoren sætter automatisk `using TMPro` ind.)

7. Sæt følgende linje ind i `Update()`:

```
scoreText.text = "Score: " + score;
```

8. I **Hierarchy**'et vælg **PlayerController** og træk **Score** over i **Score Text** feltet.

Kør spillet og se at scoren nu tælles op når man tager ting.

8 Lyd

Her vises hvordan man sætter lyd på at man tager ting. Den samme metode kan bruges til at sætte lyd på andre ting i spillet, som hop lyd, dø lyd mv. men det bliver ikke gennemgået her.

1. Lav en folder som hedder *Sounds*.
2. Find en god lyd til når man tager en ting og træk den over i **Sounds** folderen.
3. I **Hierarchy**'et vælg **Player** og tilføj componenten **Audio Source**.
4. Sæt følgende linjer ind i **Player** scriptet (de rigtige steder):

```
[SerializeField] AudioClip itemSound;
```

```
AudioSource audioSource;
```

```
audioSource = rb.GetComponent<AudioSource>();
```

5. Denne linje skal stå der, hvor man tager ting.

```
audioSource.PlayOneShot(itemSound);
```

6. Vælg **Player** og træk lydfilen over i **Item Sound** feltet.

Kør spillet og hør lyden virker, når man tager ting.

9 Hold kuglen i fart

Et af problemerne med spillet er, at når banen bliver længere så varierer kuglens fart meget og det kan give udfordringer ved hop, hvor kuglen ikke har fart nok, eller hvor den kører så hurtigt at den ikke er til at styre. Løsningen er at styre hastigheden i koden og ikke kun af, at den løber nedad (tyngdekraften). Hvis man ikke synes det er et problem, så kan man bare springe det over.

1. I **PlayerController** scriptet, tilføj følgende linjer i bunden af `Update()`:

```
if (rb.velocity.z > 10)
    rb.velocity = new Vector3(rb.velocity.x, rb.velocity.y, 10);
if (rb.velocity.z < 6)
    rb.velocity = new Vector3(rb.velocity.x, rb.velocity.y, 6);
```

Det som koden gør er at sikre at kuglens fart er større end 6 og mindre end 10 hele tiden.

10 Omgivelser (kan springes over)

Omgivelserne er vigtige for at spillet ser godt ud. Det skal ligne bygninger, men her er man nød til at bruge en anden teknik end for banen for ellers bliver der alt for mange objekter til at Unity kan køre godt. Teknikken er at man laver hver bygning som en stor klods og så lægger man et mønster på, så det ligner en bygning.

1. Tilføj følgende linjer i **RoadSpawner** (de rigtige steder):

```
using Random = UnityEngine.Random;
```

```
[SerializeField] GameObject cubeHouse;
```

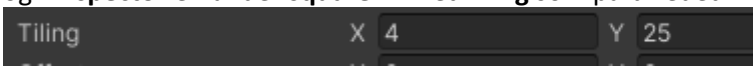
```
Vector3 envCurPos = new Vector3(0, 15, 0);
```

```
void GenEnvBuilding(float posX, float posY, float sizeX, float sizeY, float sizeZ)
{
    GameObject obj;
    Quaternion rot = Quaternion.Euler(0, 0, 0);
    obj = Instantiate(cubeHouse, envCurPos + posX * Vector3.right + posY *
Vector3.up, rot);
    obj.transform.localScale = new Vector3(sizeX, sizeY, sizeZ);
    objList.Add(obj);
}

void GenerateEnvironment()
{
    float sizeX, sizeY, sizeZ, posX, posY;
    float offX, d, off;
    Quaternion rot = Quaternion.Euler(roadCurRotationX, 0, 0);
    while (roadCurPos.z > envCurPos.z - 20)
    {
        sizeZ = 4;
        for (d = 0; d < 2; d++)
        {
            sizeX = Random.Range(3, 6);
            offX = Random.Range(0, 12);
            if (d == 0)
                posX = -5 - offX - sizeX;
            else
                posX = 5 + offX;
            off = Random.Range(-30, -12);
            posY = off;
            sizeY = off + 45;
            GenEnvBuilding(posX, posY, sizeX, sizeY, sizeZ);

            if (d == 0)
                GenEnvBuilding(posX - sizeX - 3, posY, sizeX, sizeY, sizeZ);
            else
                GenEnvBuilding(posX + sizeX + 3, posY, sizeX, sizeY, sizeZ);
        }
        envCurPos += (sizeZ + 2) * (rot * Vector3.forward);
    }
    envCurPos = new Vector3(roadCurPos.x, roadCurPos.y, envCurPos.z);
}
```

7. Tilføj linjen med **fed**:

```
if (player.transform.position.z > (roadCurPos.z - 30))
{
    ObjCleanUp();
    if (roadFuncIndex < roadFunctions.Count)
    {
        roadFunctions[roadFuncIndex]();
        roadFuncIndex++;
    }
    GenerateEnvironment();
}
```

8. Lav en ny **Cube** og kald den *CubeEnv*.
9. Find materialet til baneklodsen og kopier det og kald det *squareEnv*.
10. Træk nu squareEnv op over **CubeEnv** objektet.
11. Vælg **CubeEnv** og i **Inspector**'en under **squareEnv** ret **Tiling** som på billedet.



12. Lav en **Prefab** af den og slet den i **Hierarchy**'et.
13. Vælg **RoadSpawner** og træk **CubeEnv** over i **Cube House** feltet.

Kør spillet og se at der kommer bygninger frem.

11 Banekodningsguide

Her er en lille guide til hvordan man kan 'programmere' sin bane, så koden ikke bliver alt for lang.
Det kan være smart at lave nogle genbrugelige banesegmenter, som kan bruges flere gange.

11.1 Loops

Hvis man skal lave flere ens baneelementer efterhinanden, så skal man bruge en loop. Den kan se sådan ud:

```
for (int i = 0; i < 20; i++) {
    ObstacleAddBlock(0, 1);
    RoadAddBlock(RB_A);
}
```

Her laver man 20 baneelementer og på hvert element står der en forhindring.

11.2 Banehældning

Her laver man et stykke bane som har en hældning. Funktionen tager et argument og det er hældningen. Den laver også en lille afstand så man naturligt kan komme "over på" det nye stykke bane selvom det hælder.

1. Tilføj følgende funktion til **RoadSpawner**:

```
void RoadSegmentAngle(float angle)
{
    RoadAddSpace(-3, 2);
    for (int i = 0; i < 20; i++)
        RoadAddBlock(RB_A, angle);
}
```

Vi kan nu bruge den nye funktion flere gange til at lave en bane som første hælder den ene vej, så vandret og til sidst hælder den anden vej.

2. Tilføj nu følgende linjer i `Start()`:

```
roadFunctions.Add(() => RoadSegmentAngle(-20));
roadFunctions.Add(() => RoadSegmentAngle(0));
roadFunctions.Add(() => RoadSegmentAngle(20));
```

Kør koden og se hvordan det virker.

11.3 Betingelser

Det kan også være smart at kunne genbruge den samme kode, men hvor der er små varianter, fx om der er forhindringer eller ej. Til det formål bruger man en `if` sætning.
Den følgende funktion laver et stykke bane, men hvor man kan vælge om den skal have forhindringer eller ej.
Tilføj følgende funktion til **RoadSpawner** scriptet:

```
    void RoadSegmentObs(bool obs)
    {
        for (int i = 0; i < 10; i++)
        {
            if (obs)
                ObstacleAddBlock(-1, 1);
            RoadAddBlock(RB_A);
        }
    }
```

Parameteren `obs` afgør om den skal lave en forhindring eller ej.

Tilføj følgende linjer til i `Start()`:

```
roadFunctions.Add(() => RoadSegmentObs(false));
roadFunctions.Add(() => RoadSegmentObs(true));
```

Kør spillet og se forskellen på de to stykker bane.

11.4 Tilfældighed

Man kan også lave banen, så den er lidt tilfældig, fx hvor stor en hældning er eller om der er forhindringer eller ej.
Til det formål bruger vi en funktion som hedder Random som kan lave et tilfældigt tal.

0. Hvis du ikke har lavet *Omgivelser* kapitlet, så skal du start med at tilføje følgende linje:

```
using Random = UnityEngine.Random;
```

Funktionen `Random.Range(min, max)` leverer et heltal tilbage i intervallet **[min, max[**, det vil sige at tallet er mindst **min**, og maksimalt **max-1**.
Eksempel:

```
x = Random.Range(2, 7);
```

Her kan `x` få følgende værdier: {2, 3, 4, 5, 6}.
Følgende kode laver en lille 'labyrint', hvor forhindringerne står tilfældigt. Læg mærke til at den bruger to loops inden i hinanden for at genbruge så meget kode som muligt.

1. Tilføj følgende linjer til **RoadSpawner** scriptet:

```
void RoadSegmentRandom()
{
    int x;
    for (int j = 0; j < 5; j++)
    {
        x = Random.Range(-1, 2);
        ObstacleAddBlock(x, 1);
        for (int i = 0; i < 7; i++)
            RoadAddBlock(RB_A);
    }
}
```

I dette tilfælde er x = {-1, 0, 1}.

2. Tilføj følgende linje til `Start()`:

```
roadFunctions.Add(() => RoadSegmentRandom());
```

Kør spillet og prøv "labyrinten".

11.5 Uendelig – genbrug segmenter

Banen skal jo forsætte i det uendelige og det kræver jo mange baner 😉. Løsningen er at genbruge banesegmenterne så banen på den måde bliver ved for evigt.

1. I **RoadSpawner** scriptet tilføj de linjer med **fed** i `Update()` funktionen.

```
if (roadFuncIndex < roadFunctions.Count)
{
    roadFunctions[roadFuncIndex]();
    roadFuncIndex++;
}
else
{
    roadFuncIndex = 2;
}
```

Linjen `roadFuncIndex = 2`, betyder at man starter 'forfra' på banesegment 2 og fremad og derfor bliver spillet aldrig færdigt.

12 Next steps

- Tunneller
- Power-ups
- Start menu
- Level counter

13 Links

Tips videoer	TOP 10 UNITY TIPS - 2017 TOP 10 UNITY TIPS #2
Unity manual	https://docs.unity3d.com/2022.3/Documentation/ScriptReference/index.html
Unity assets store	https://assetstore.unity.com
Dansk C#/Unity document	https://github.com/Grailas/CodingPiratesAalborg/blob/master/Guides/Hj%C3%A6lpeguide.pdf