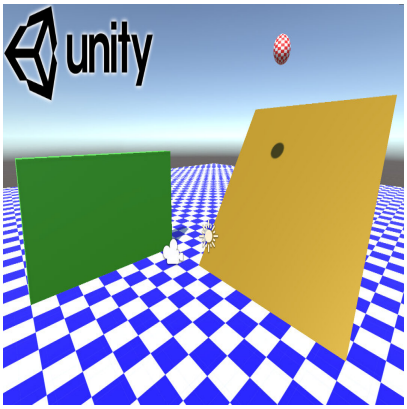


Kamera verfolgt den Ball

Description

Dieses Tutorial baut auf dem Ball-Tutorial auf. Wir erweitern die Szene, indem wir die Kamera dem Ball folgen lassen. Außerdem bauen wir noch eine Rampe und eine Wand ein, um den Ball besser testen zu können.

Das ist das Ergebnis:

https://www.bytegame.de/wp-content/uploads/2024/01/Ball-02.mp4

Wie man sehen kann, hat sich die Verwendung der Bodentextur gelohnt, weil man so sehr gut erkennen kann, dass die Kamera dem Ball folgt. Und hier fangen wir gleich an.

CameraFollowsObject

Es ist zweckmäßig, solche Skripte allgemein zu schreiben. Es ist somit egal, ob die Kamera einem Ball, Würfel, Wachmann oder Zug folgt. Das Prinzip ist in Unity sehr simpel.

```
using UnityEngine;

public class CameraFollowsObject : MonoBehaviour
{
    public Transform followObject; // Das Objekt, dem die Kamera folgen soll
    public Vector3 offset = new Vector3(0f, 3f, -2f); // Offset, um die Kamera

    void Update()
    {
        if (followObject != null)
        {
            // Berechne die Kameraposition
            Vector3 targetPosition = followObject.position + offset;

            // Interpoliere sanft zwischen der aktuellen Kameraposition und de
            transform.position = Vector3.Lerp(transform.position, targetPositi
```

```
        }
    }
}
```

Das ist, wenn man so will, die Basisfunktion. Darauf aufbauend könnte man es bspw. so abändern, dass man per Mausrad noch den Abstand beeinflussen kann oder man generell den Blickwinkel verändert. Dazu wird es bestimmt irgendwann noch ein Tutorial geben.

Dieses Skript müssen wir nur noch der Kamera zuweisen und die drei Abstände zum Ball unseren Wünschen anpassen. Fertig.

Hindernisse und Kollision

Es ist zugegebenermaßen nicht sonderlich aufregend, einen Ball auf einer ebenen Fläche rollen zu sehen. Also bauen wir ein paar Hindernisse ein. Ich habe mich mit einer Rampe und einer Wand begnügt, du darfst hier gerne kreativer sein.

Die Rampe habe ich, sie im Video zu sehen ist, um 45° geneigt. Setzt man, wie im Titelbild des Artikels zu sehen, den Ball über die Rampe, fällt dieser nach Spielstart herunter und rollt bis zum Boden. Wir erkennen hier auch, dass die Kamera ein bisschen braucht, bis sie zum Ball fliegt.

Die Rampe ist, wie der Boden, eine Plane. Wenn wir mit dem Ball hinter die Rampe rollen, können wir nach vorne hindurchrollen. Das heißt, dass wir hier ein Kollisionsproblem haben. Dafür brauchen wir einen *Mesh Collider* … aber halt, der ist ja schon da! Warum funktioniert das nicht?

Das liegt daran, dass der Schalter *Convex* standardmäßig aus ist. Den müssen wir einschalten, wenn wir auch eine Kollision von der Rückseite verhindern möchten.

Erklärung Convex

Ein konvexes Mesh ist ein Mesh, bei dem jede Verbindungslinie zwischen zwei Punkten innerhalb des Meshs komplett innerhalb des Meshs liegt.

Der Begriff „konvex” bezieht sich auf die Geometrie des Meshs. Eine konvexe Form ist eine, bei der eine gerade Linie zwischen zwei Punkten innerhalb der Form bleibt, ohne die Grenzen der Form zu durchdringen. Im Gegensatz dazu wäre eine konkave Form eine, bei der es möglich ist, eine gerade Linie zwischen zwei Punkten zu zeichnen, die die Grenzen der Form durchschneidet.

Der Unity *Mesh Collider* kann nur mit konvexen Meshes verwendet werden. Wenn das Mesh nicht konvex ist, gibt es zwei Möglichkeiten:

1. **Manuell konvexes Mesh erstellen:** Du kannst das Mesh manuell so bearbeiten, dass es konvex wird. Dies kann jedoch in komplexen Szenarien schwierig sein und die Genauigkeit der ursprünglichen Form beeinträchtigen.

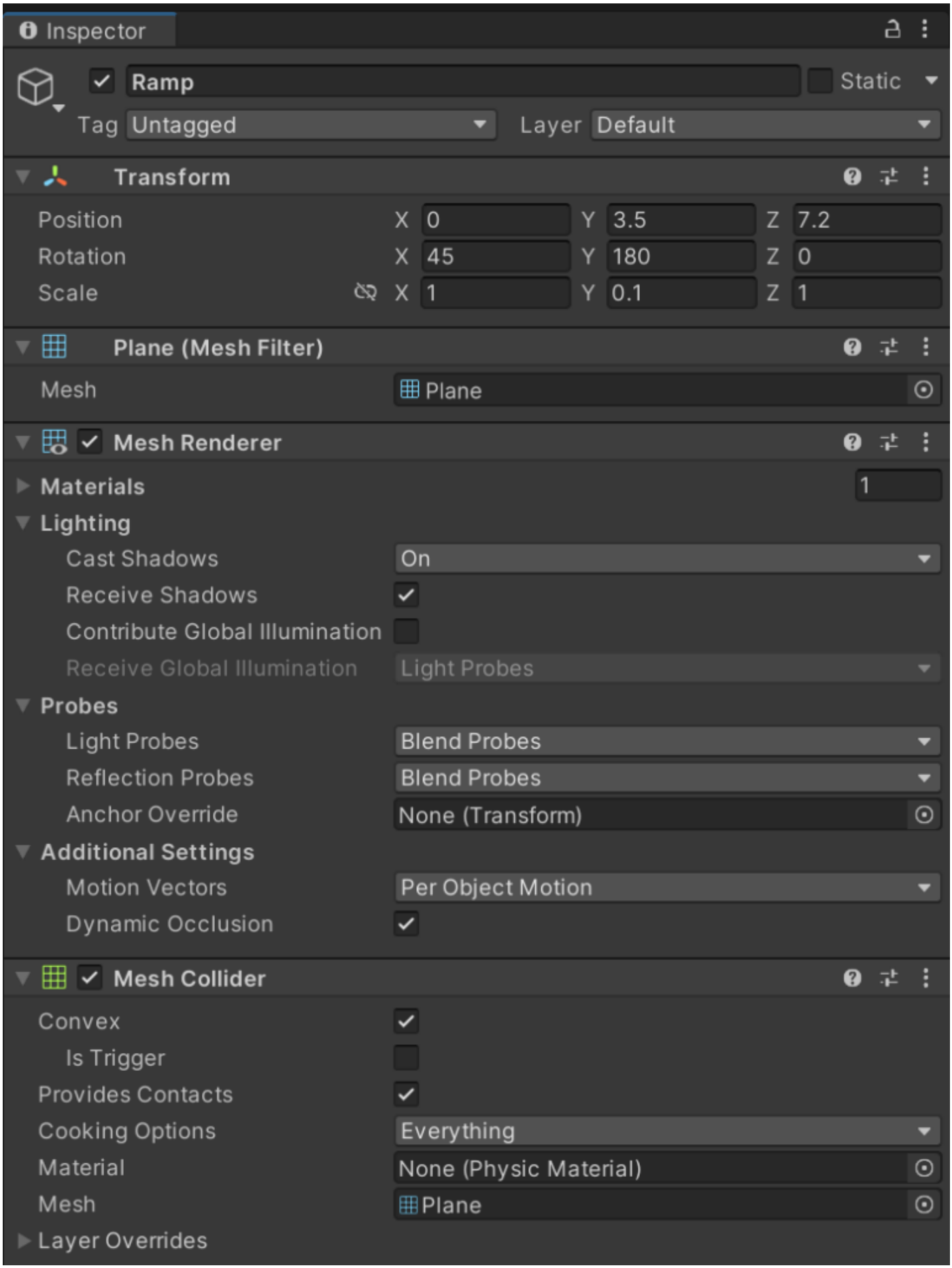
2.

Mehrere Mesh Collider verwenden: Alternativ kannst du das Mesh in mehrere Teile aufteilen, um separate Mesh Collider-Komponenten zu erstellen, die jeweils konvex sind. Beachte, dass dies mehr Ressourcen verbrauchen kann und möglicherweise nicht in allen Szenarien ideal ist.

Es ist wichtig zu beachten, dass die Einstellung *Convex* für den Mesh Collider die Leistung beeinflussen kann. Konvexe Mesh Collider sind in der Regel schneller zu verarbeiten als nicht-konvexe, aber sie können nicht alle Arten von Meshes genau repräsentieren. Daher ist es wichtig, die Anforderungen der spezifischen Anwendung zu berücksichtigen und die beste Lösung für die Kollisionsanforderungen zu wählen.

Der Ball schwebt?

Ja, wenn man mit der Kamera von der Seite darauf schaut, stellt man fest, dass der Ball nicht auf der Fläche liegt, sondern schwebt. Die Kollisionsbox ist wesentlich dicker als die Fläche. Die Erklärung ist einfach: Wenn man eine *Plane* erstellt, hat sie die Maße 1, 1, 1. Y ist dabei die Dicke. Eine *Plane* hat aber keine Dicke, wir können sie aber auch nicht auf Null stellen. Lösung: Wir stellen sie auf 0,1. Insgesamt sieht das bei mir so aus:


Inspector
Ramp
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 3.5 Z 7.2
Rotation X 45 Y 180 Z 0
Scale X 1 Y 0.1 Z 1
Plane (Mesh Filter)
Mesh Plane
Mesh Renderer
Materials 1
Lighting
Cast Shadows On
Receive Shadows
Contribute Global Illumination
Receive Global Illumination Light Probes
Probes
Light Probes Blend Probes
Reflection Probes Blend Probes
Anchor Override None (Transform)
Additional Settings
Motion Vectors Per Object Motion
Dynamic Occlusion
Mesh Collider
Convex
Is Trigger
Provides Contacts
Cooking Options Everything
Material None (Physic Material)
Mesh Plane
Layer Overrides

Einstellungen der Rampe

Lektionen

Das war eigentlich nicht schwierig, aber wir haben einiges gelernt.

1. Kollisionsabfragen sind rechenintensiv.
2. Kollisionen muss man im Spiel ordentlich prüfen. Nicht umsonst gibt es gerade hier selbst in professionellen Spielen sehr viele Bugs.
3. Selbst vermeintlich einfache Dinge haben ihre Tücken.

Nun kannst du mit den Werten experimentieren und die Welt ein wenig ausbauen. Viel Spaß damit!

Weiterführende Links

Unity – Einstieg in die Spieleentwicklung
Einstieg in Unity Teil 1 – 3D und erste Szene
Legenden – John Carmack

Date Created
19. Januar 2024
Author
sven