Der rollende Ball

Description

In diesem Tutorial wirst du lernen, wie man in Unity einen Ball zum rollen bringt und ihn mit W-A-S-D und den Pfeiltasten steuert.

Für Anfänger

Dieses Tutorial ist für Anfänger. Wer allerdings noch nie etwas mit der Unity-Engine zu tun hatte, sollte sich zunächst die dreiteilige Einsteigerserie anschauen. Dieses Tutorial baut auf dem Wissen auf.

Zu beginn werden wir eine kleine Szene basteln und uns dann dem Code widmen. Dabei werde ich alles sehr einfach und übersichtlich handhaben. Auch wenn wir ein wenig mit Physik arbeiten werden, wird es ziemlich easy, da uns Unity sehr viele Dinge abnimmt.

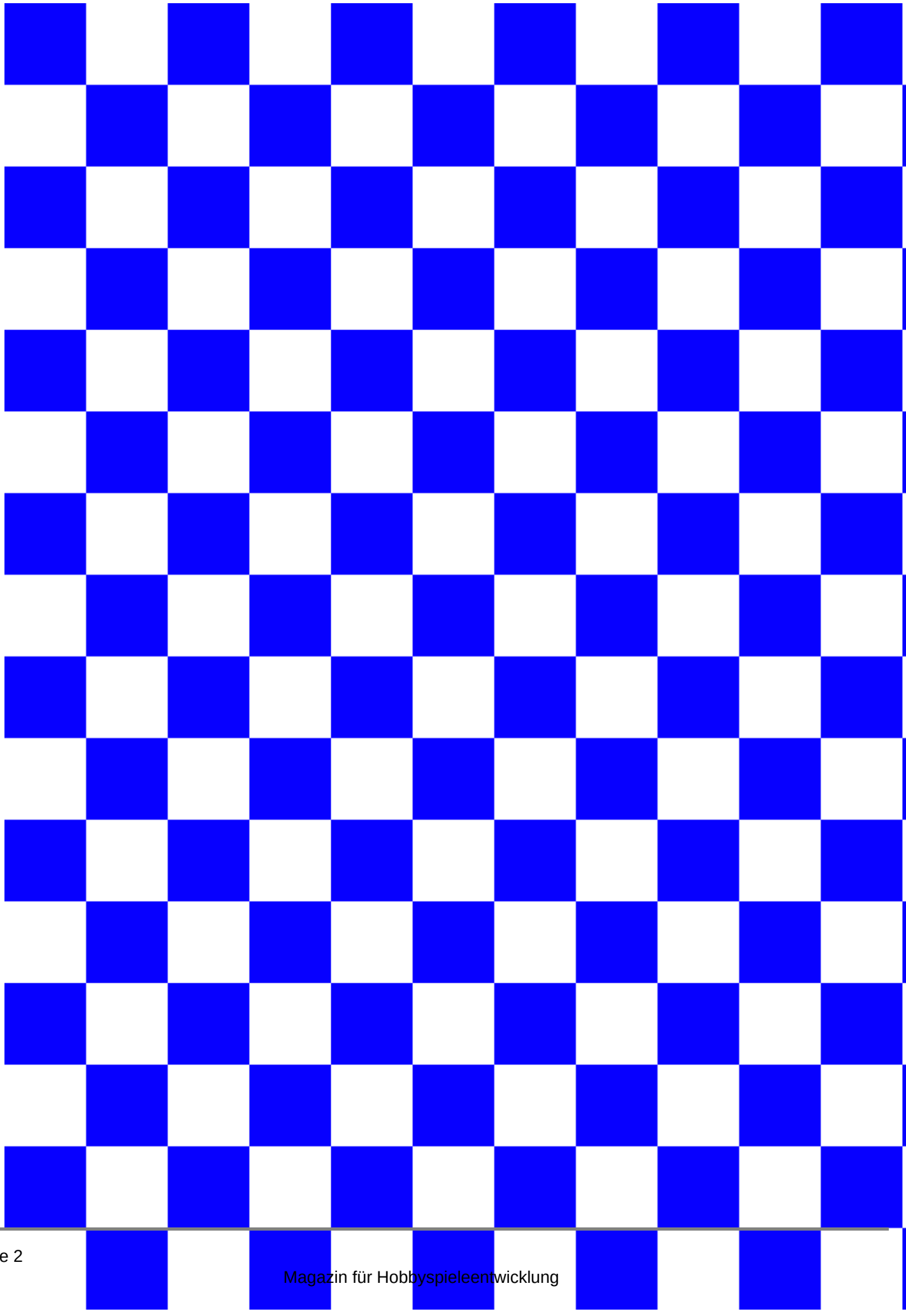
Am Ende sollte die Szene so aussehen:

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

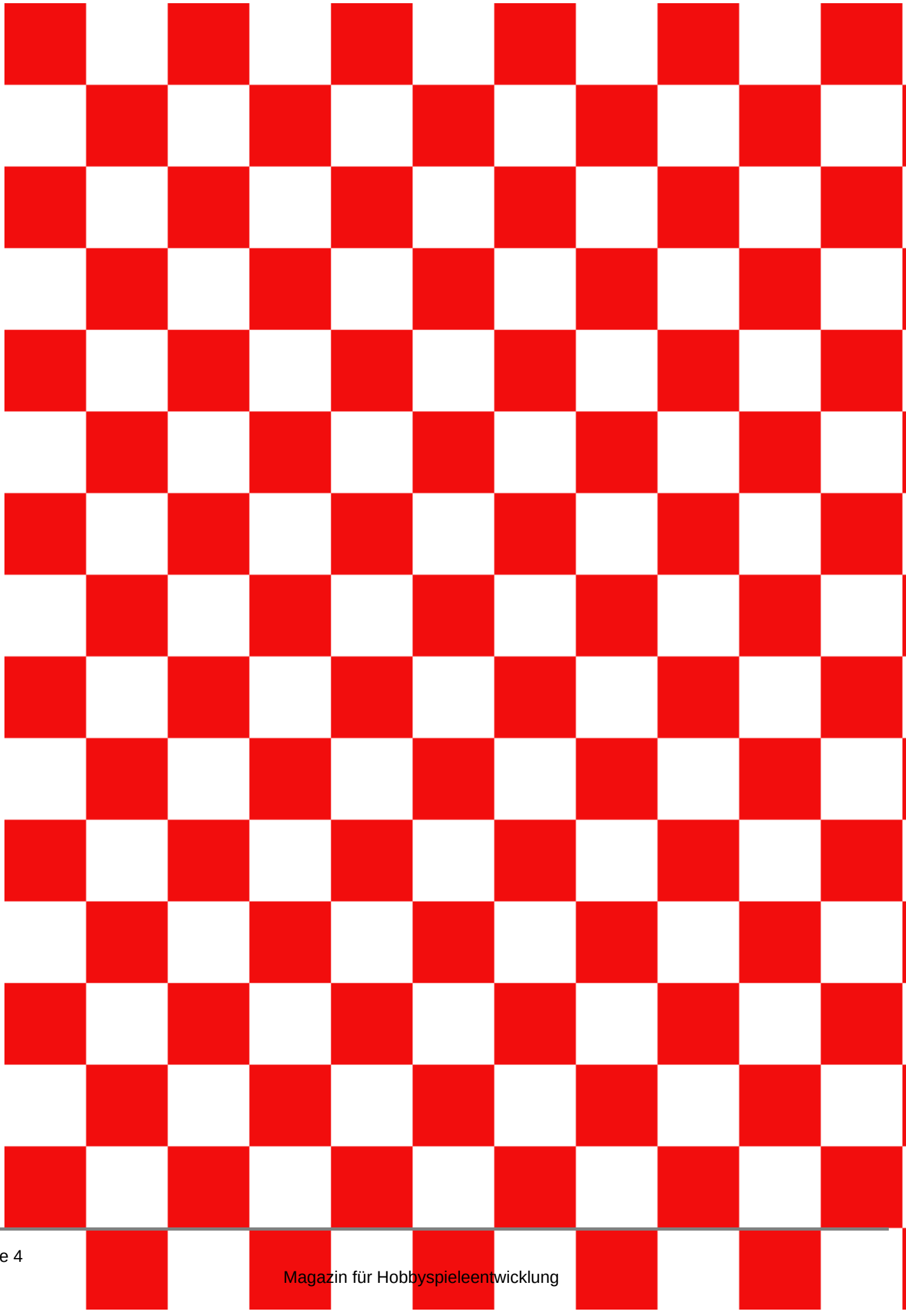
Zwischendurch werde ich einige Dinge tiefergehend erklären. Du kannst das aber gerne überspringen und zu einem späteren Zeitpunkt durchlesen.

Die Szene

Wir starten mit einer *3D Basic* Szene. Darin haben wir bereits Kamera, Licht und einen Himmel. Dem fügen wir, wie in der Einsteigerserie gezeigt, einen Boden (Plane) und einen Ball (Sphere) hinzu. Für Boden und Ball erzeugen wir noch je ein Material und weisen eine Textur zu. Vor allem beim Ball ist es nützlich, damit wir die Drehung erkennen. Du kannst hierzu gerne meine Texturen verwenden.

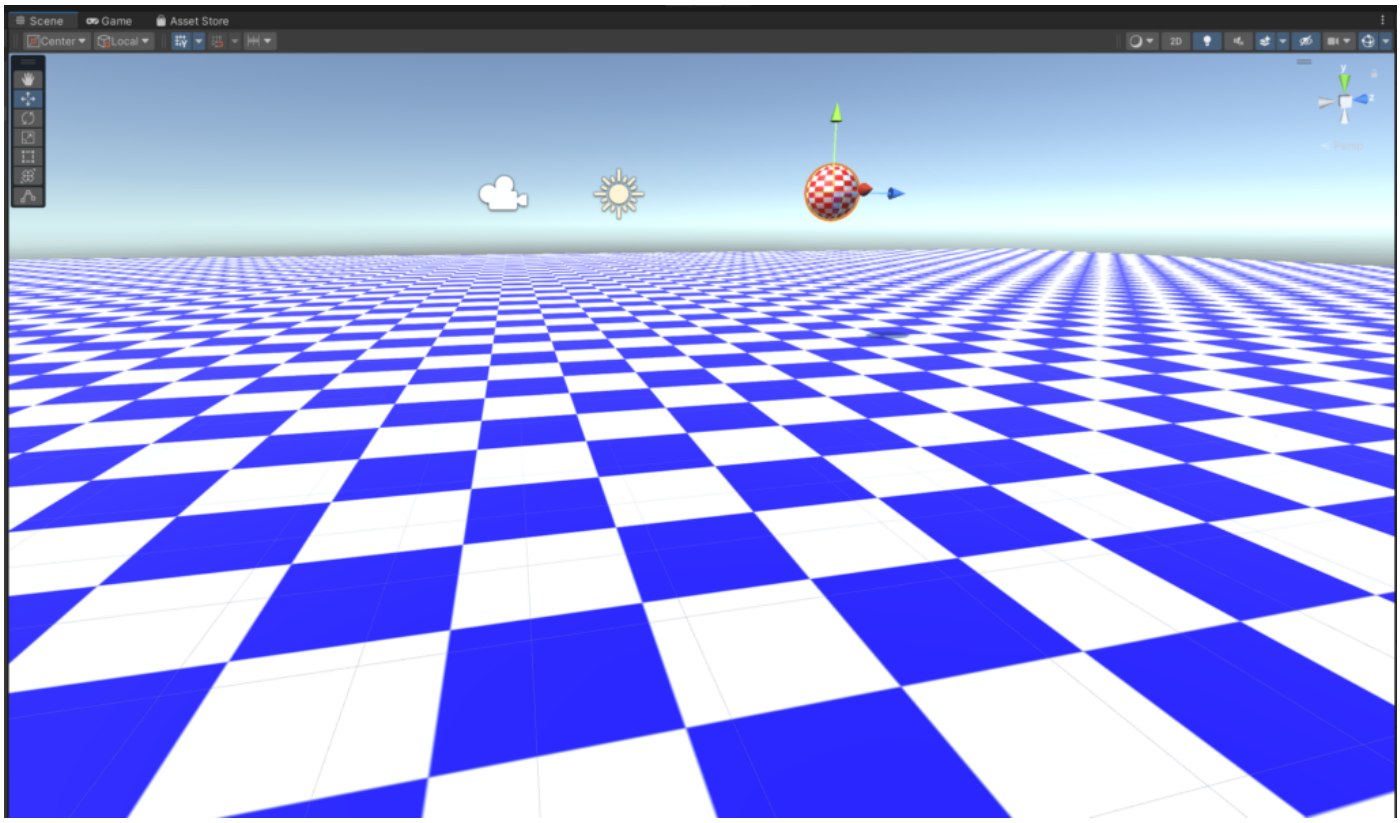


Bodentextur



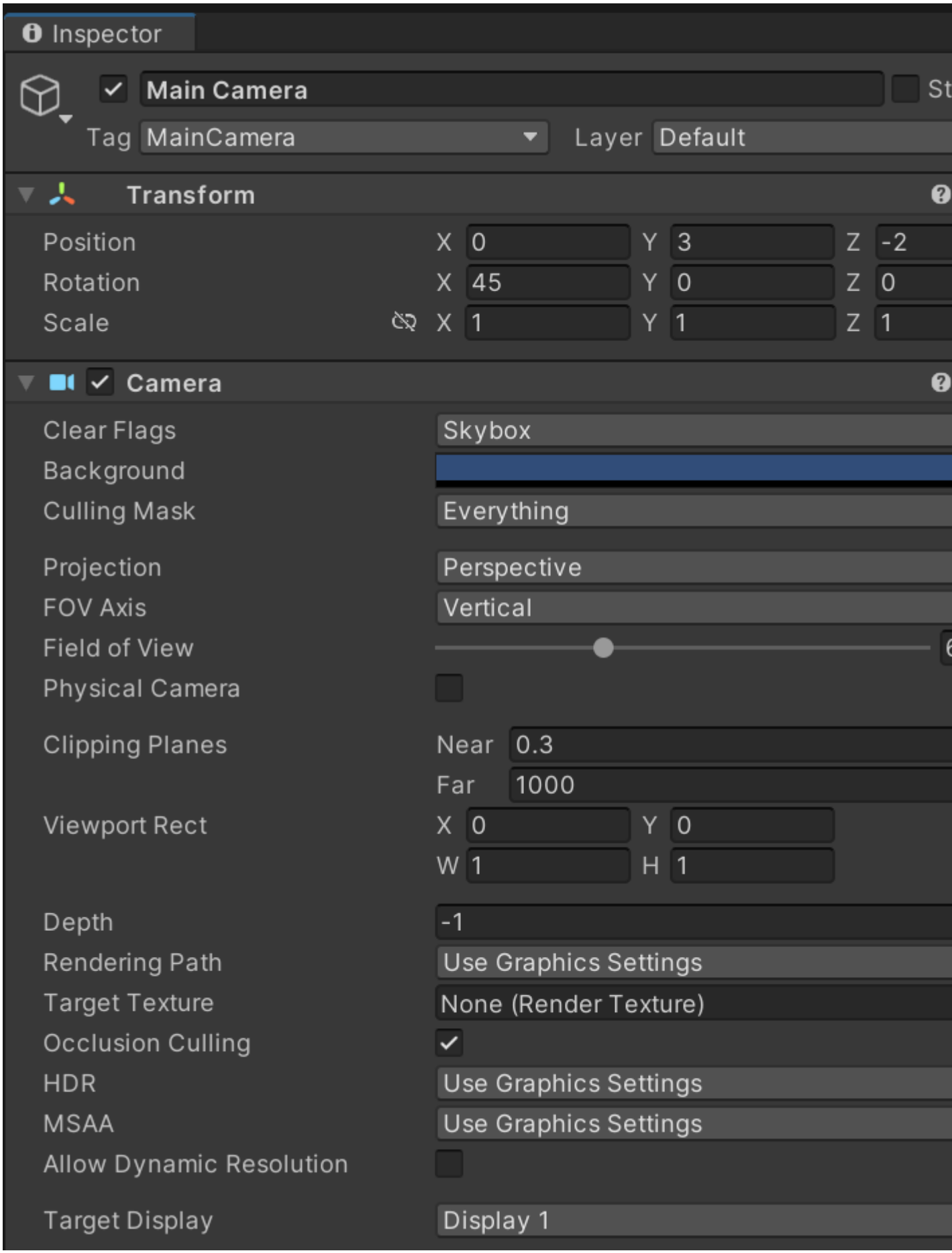
Textur für den Ball

Setze den Ball üben den Boden. Durch unsere Einstellungen wird er zu beginn der Szene herunterfallen und auf dem Boden landen. Bei mir sieht das so aus:



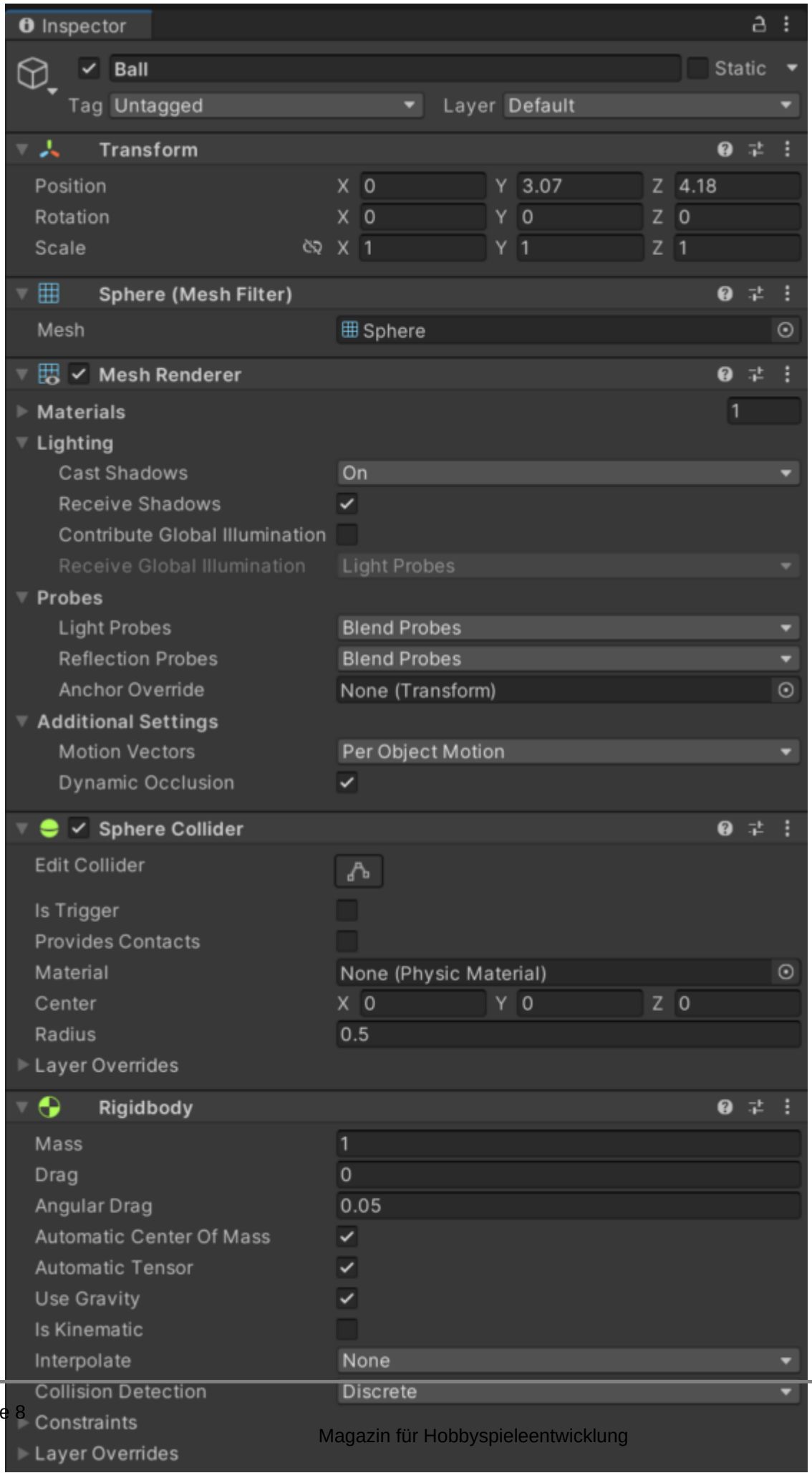
Szene im Unity-Editor

Die Kamera solltest du auf der x-Achse um 45° drehen. Bei mir sehen die Einstellungen für die Kamera so aus:


Inspector
Main Camera
Tag MainCamera
Layer Default
Transform
Position X 0 Y 3 Z -2
Rotation X 45 Y 0 Z 0
Scale X 1 Y 1 Z 1
Camera
Clear Flags Skybox
Background
Culling Mask Everything
Projection Perspective
FOV Axis Vertical
Field of View
Physical Camera
Clipping Planes Near 0.3
Far 1000
Viewport Rect X 0 Y 0
W 1 H 1
Depth -1
Rendering Path Use Graphics Settings
Target Texture None (Render Texture)
Occlusion Culling
HDR Use Graphics Settings
MSAA Use Graphics Settings
Allow Dynamic Resolution
Target Display Display 1

Camera Inspector

Dem Ball fügst du einen einen *Sphere Collider* hinzu. Den brauchst du für die Kollision. Außerdem noch ein *Rigidbody*. Die Einstellungen sehen bei mir so aus:


Inspector
Ball
Static
Tag Untagged
Layer Default
Transform
Position X 0 Y 3.07 Z 4.18
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1
Sphere (Mesh Filter)
Mesh Sphere
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
Sphere Collider
Edit Collider
Is Trigger
Provides Contacts
Material None (Physic Material)
Center X 0 Y 0 Z 0
Radius 0.5
Layer Overrides
Rigidbody
Mass 1
Drag 0
Angular Drag 0.05
Automatic Center Of Mass
Automatic Tensor
Use Gravity
Is Kinematic
Interpolate None
Collision Detection Discrete
Constraints
Layer Overrides

Ball Inspector

Erklärungen

Sphere Collider

Dies ist eine Kollisionskomponente, die eine sphärische Form repräsentiert. Ein *Collider* in Unity wird verwendet, um festzustellen, ob Objekte miteinander kollidieren. Der *Sphere Collider* ist speziell darauf ausgelegt, Kollisionen basierend auf einer Kugelform zu erkennen.

Wenn du einen *Sphere Collider* zu einem Spielobjekt hinzufügst, wird dieses Spielobjekt eine unsichtbare sphärische Hülle um sich haben. Diese Hülle wird für Kollisionen mit anderen Objekten in der Spielwelt verwendet. Wenn sich die *Collider* zweier Objekte überlappen, wird eine Kollision erkannt, und du kannst dann entsprechend darauf reagieren, indem du beispielsweise Skripte für Kollisionsereignisse verwendest.

Sphere Collider sind besonders nützlich, wenn du Kollisionen in Spielsituationen modellieren möchtest, bei denen eine sphärische Form besser geeignet ist als andere Collider-Typen wie *Box Collider* oder *Mesh Collider*. Collider sind nur dann effektiv sind, wenn sie mit einem *Rigidbody*-Komponente kombiniert werden.

Rigidbody

Rigidbody ist eine Komponente, die einem Spielobjekt physikalische Eigenschaften verleiht.
Damit kann ein Objekt den Gesetzen der Physik gehorchen und reagieren, zum Beispiel auf Kräfte, Schwerkraft, Kollisionen und Drehmomente.

Außerdem ist auch Interpolation und Extrapolation möglich. Diese Funktionen helfen, die Bewegung von Objekten zwischen zwei Frames zu glätten, was insbesondere bei schnellen oder unregelmäßigen Bewegungen wichtig sein kann.

Um einen *Rigidbody* zu einem Spielobjekt hinzuzufügen, kannst du dies entweder über den Unity-Editor tun oder programmatisch in einem Skript. Beachte, dass du in Unity oft auf Kombinationen von *Rigidbody* und *Collidern* angewiesen bist, um realistische physikalische Interaktionen in deinem Spiel zu erreichen.

Skript BallController

Eigentlich ist das Skript ziemlich kurz, es wird lediglich durch die Kommentare etwas aufgebläht, also keine Panik.

```
using UnityEngine;

// Dieses Skript erfordert ein Rigidbody-Komponente, um korrekt zu funktionier
[RequireComponent(typeof(Rigidbody))]
public class BallController : MonoBehaviour
```

```
{
    // Die Kraft, um den Ball zu bewegen
    public float moveForce = 100f;

    // Das Drehmoment, um den Ball zu drehen
    public float rotateTorque = 100f;

    // Die Referenz auf das Rigidbody-Objekt
    private Rigidbody rb;

    // Wird einmal beim Start des Spiels aufgerufen
    void Start()
    {
        // Holen die Referenz auf das Rigidbody-Objekt, das mit diesem Skript
        rb = GetComponent<Rigidbody>();

        // Setze den angularDrag-Wert auf 0.5f, um die Drehbewegung zu dämpfen
        rb.angularDrag = 0.5f;
    }

    // Wird in jedem Frame aufgerufen
    void Update()
    {
        // Bewegungseingaben des Spielers abrufen
        float horizontalInput = Input.GetAxis("Horizontal");
        float verticalInput = Input.GetAxis("Vertical");

        // Die Bewegungsrichtung normalisieren, um sicherzustellen, dass die D
        Vector3 moveDirection = new Vector3(horizontalInput, 0f, verticalInput

        // Rotation des Balls, wenn er sich bewegt
        if (moveDirection != Vector3.zero)
        {
            // Das Drehmoment um die y-Achse basierend auf der Bewegungsrichtu
            Vector3 torque = new Vector3(moveDirection.z, 0f, -moveDirection.x

            // Das Drehmoment auf den Rigidbody anwenden
            rb.AddTorque(torque);
        }

        // Bewegung des Balls auf der Fläche
        if (Input.GetButtonDown("Vertical") || Input.GetButtonDown("Horizontal
        {
            // Die Kraft berechnen, die auf den Ball angewendet wird, basieren
            Vector3 moveForceVector = moveDirection * moveForce;

            // Die aktuelle y-Geschwindigkeit beibehalten, um die Bewegung in
            moveForceVector.y = rb.velocity.y;

            // Die Kraft auf den Rigidbody anwenden
            rb.AddForce(moveForceVector);
        }
    }
```

```
}
```

Erklärungen

Erforderliche Komponente

Das Skript erfordert, dass das Spielobjekt über eine `Rigidbody`-Komponente verfügt. Dies wird durch die `[RequireComponent(typeof(Rigidbody))]` Zeile am Anfang des Skripts sichergestellt.

Variablen

`moveForce`: Die Kraft, die auf den Ball angewendet wird, um ihn zu bewegen.

`rotateTorque`: Das Drehmoment, das auf den Ball angewendet wird, um ihn zu drehen.

`rb`: Eine private Referenz auf die `Rigidbody`-Komponente des Spielobjekts.

Start-Methode

Die `Start`-Methode wird einmal beim Start des Spiels aufgerufen.

Die Referenz auf die `Rigidbody`-Komponente wird abgerufen, und der `angularDrag` wird auf 0.5f gesetzt, um die Drehbewegung zu dämpfen.

Update-Methode

Die `Update`-Methode wird in jedem Frame aufgerufen.

Die horizontale und vertikale Eingabe des Spielers wird über `Input.GetAxis` abgerufen.

Die Eingaben werden normalisiert, um sicherzustellen, dass diagonale Bewegungen nicht schneller sind.

Die Rotation des Balls wird basierend auf der Bewegungsrichtung berechnet und auf den Ball angewendet, wenn er sich bewegt.

Bewegung des Balls

Wenn der Spieler die Bewegungstasten drückt (`Input.GetButtonDown("Vertical")` oder `Input.GetButtonDown("Horizontal")`), wird die Kraft berechnet, die auf den Ball angewendet wird.

Die aktuelle y-Geschwindigkeit des Balls wird beibehalten, um die vertikale Bewegung zu ignorieren.

Die berechnete Kraft wird dann auf den `Rigidbody` angewendet, um den Ball zu bewegen.

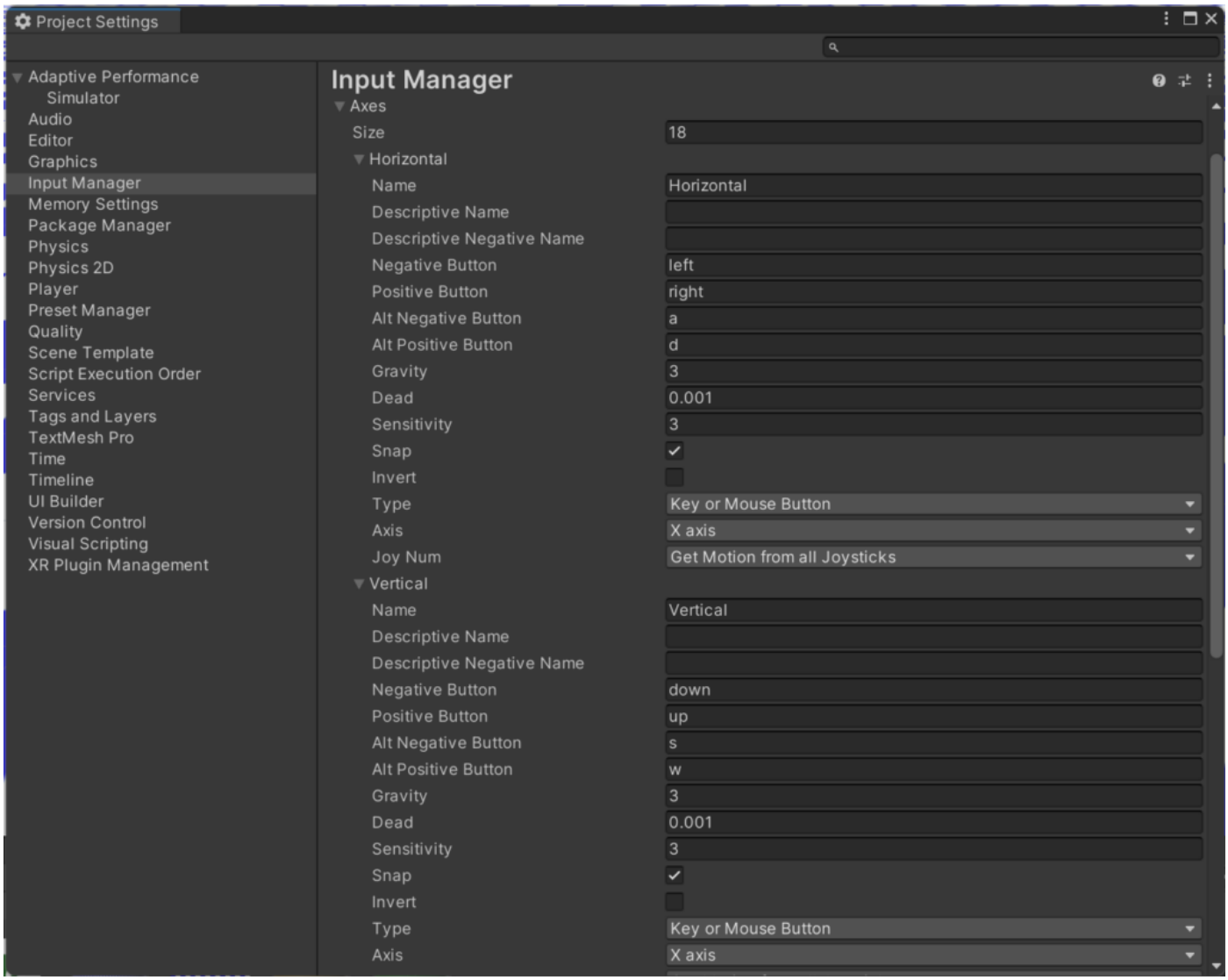
Die Steuerung erfolgt also indirekt über die Bewegungsrichtung des Spielers. Die Tasten „Vertical” und

„Horizontal” werden verwendet, um die horizontale und vertikale Eingabe des Spielers zu erfassen, und die Bewegung des Balls wird dann durch die Anwendung von Kräften und Drehmomenten auf den `Rigidbody` gesteuert. Wenn du das Skript in deinem Unity-Projekt verwendest, kannst du die Pfeiltasten oder die W-A-S-D-Tasten verwenden, um den Ball zu steuern.

Im vorliegenden Skript wird die Eingabe über die Funktion `Input.GetAxis("Horizontal")` für die horizontale Richtung und `Input.GetAxis("Vertical")` für die vertikale Richtung abgerufen. Diese Funktionen in Unity sind standardmäßig so konfiguriert, dass sie auf die Pfeiltasten und die W-A-S-D-Tasten reagieren.

Die `Input.GetAxis`-Funktion ist darauf ausgelegt, sowohl auf die Pfeiltasten als auch auf die W-A-S-D-Tasten zu reagieren, ohne dass dies speziell im Skript festgelegt werden muss. Unity interpretiert diese Tasten standardmäßig als horizontale und vertikale Eingabe. Wenn du die Pfeiltasten oder W-A-S-D-Tasten auf deiner Tastatur verwendest, reagiert das Skript automatisch auf diese Eingaben.

Diese Einstellungen können auch in den Unity-Eingabeoptionen angepasst werden. Du kannst dies überprüfen, indem du zu „**Edit ? Project Settings ? Input Manager**” gehst und die Achsen *Horizontal* und *Vertical* überprüfst. Dort siehst du die zugewiesenen Tasten und kannst sie bei Bedarf ändern.



Einstellungen der Steuerung in Unity

Was fehlt noch?

Du wist schnell bemerken, dass der Ball aus dem Sichtfeld verschwindet, wenn er etwas zu weit rollt. Es wäre also hilfreich, wenn die Kamera dem Ball folgen würde. Darum kümmern wir uns im nächsten Tutorial.

Außerdem bauen wir eine Rampe und eine Wand ein, um die Kollision zu testen.

Weiterführende Links

Unity – Einstieg in die Spieleentwicklung
Lernkurven in Spielen
Die hohe Kunst des Rätseldesigns
Das Element der Überraschung

Date Created
12. Januar 2024

Author
sven