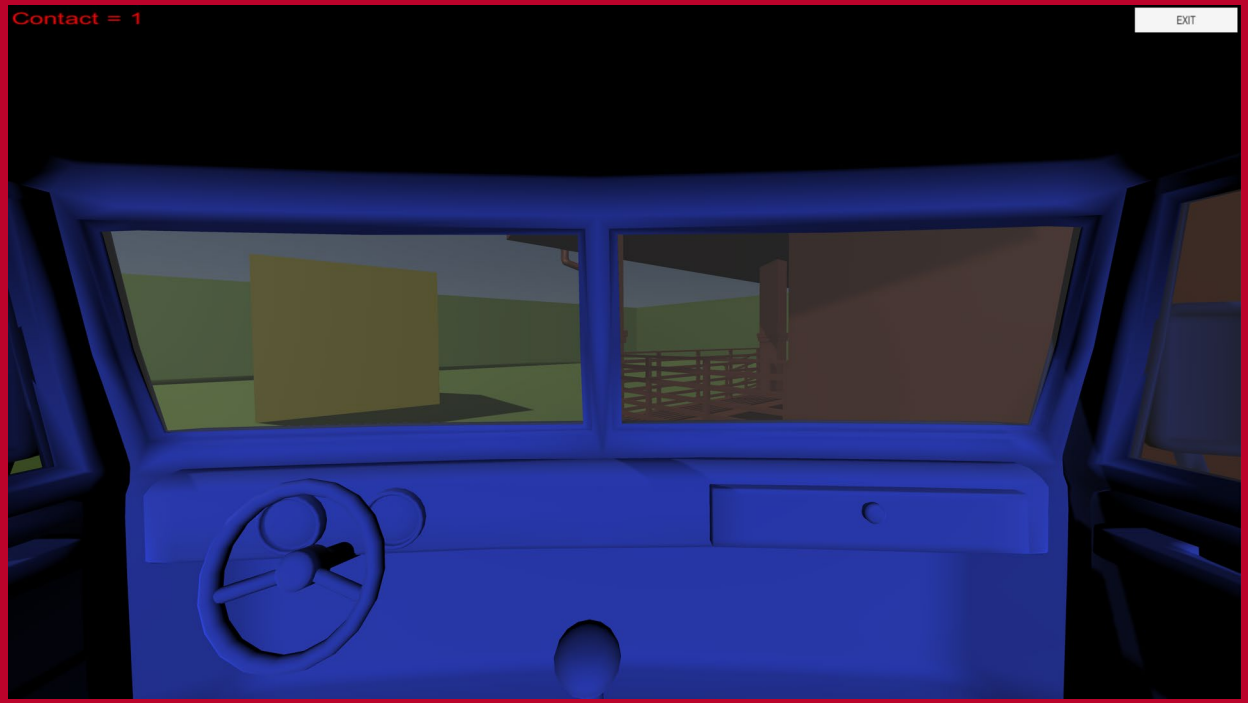
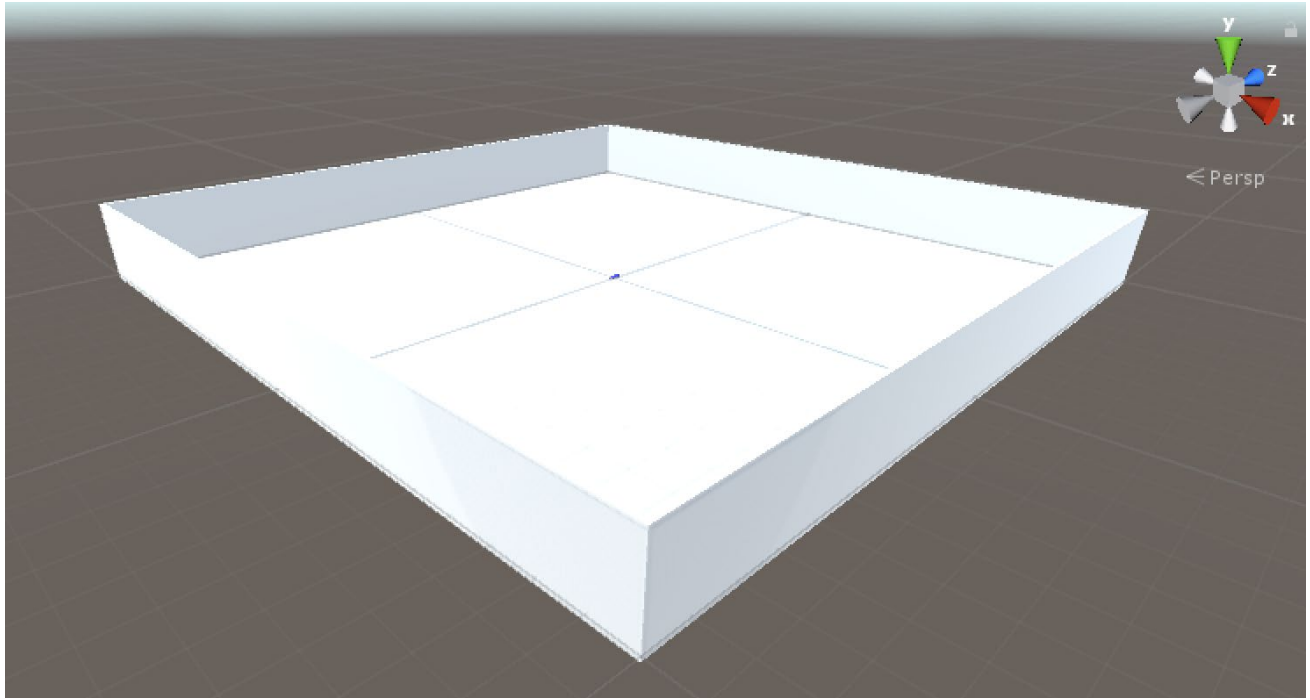
Unity 2019.16.1f1

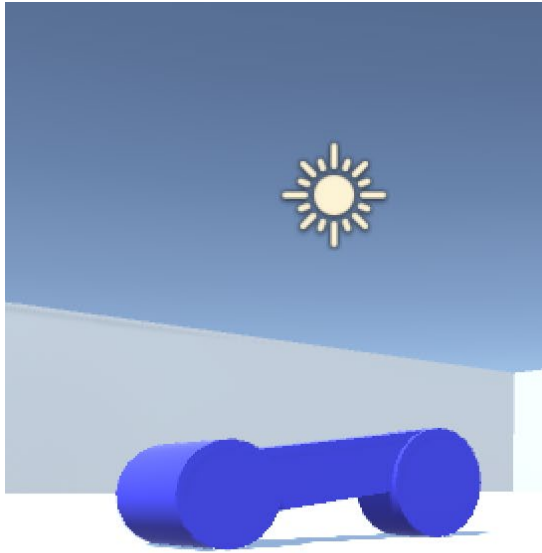
Erweiterung der Übung Collision Detection



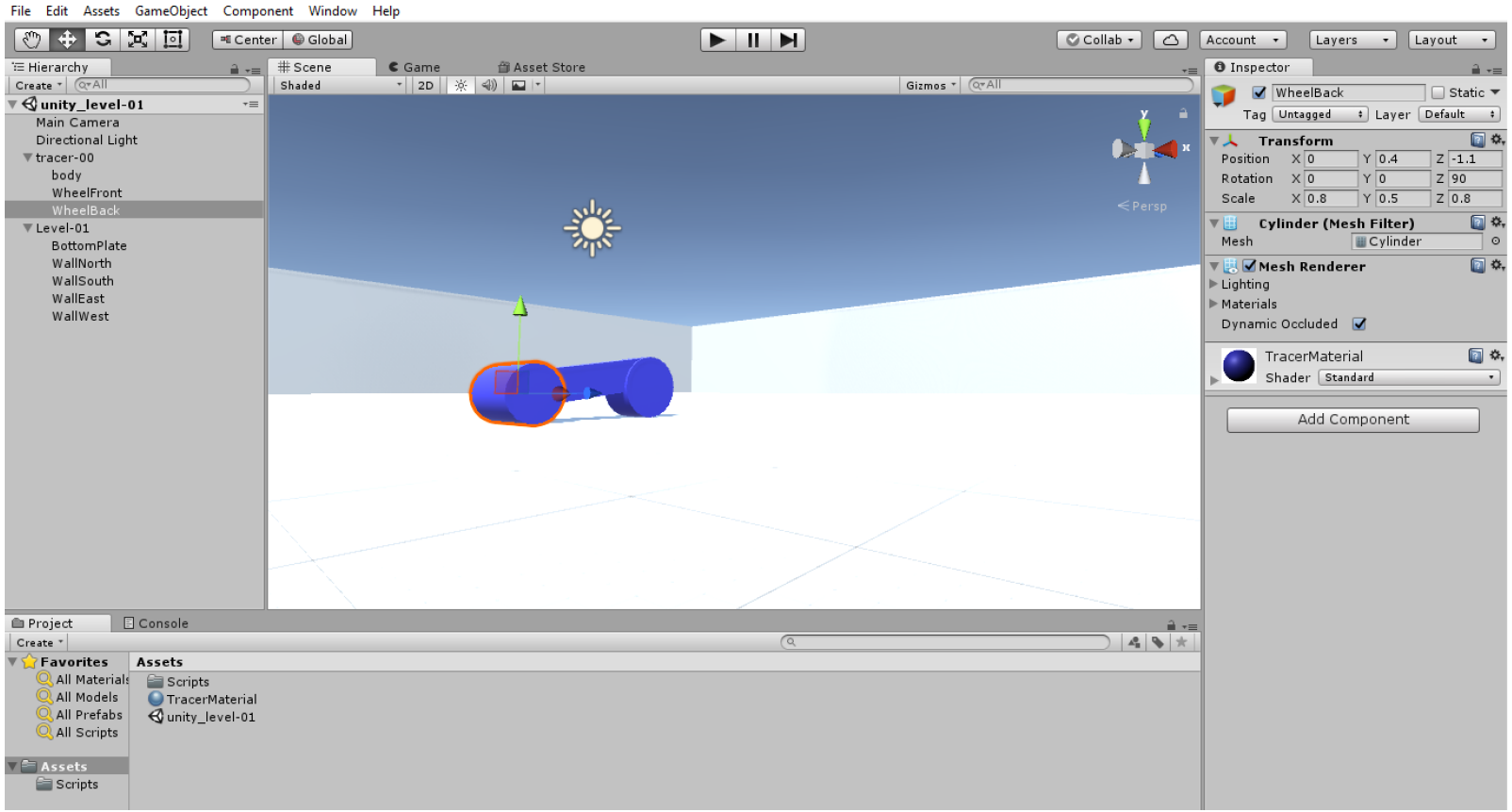
Ziel

Ziel dieses Tutorials ist die Erstellung eines einfachen 3D-Spiels mit einem simplen Fahrzeug und einer Fläche, die mit vier Wänden umgeben ist.



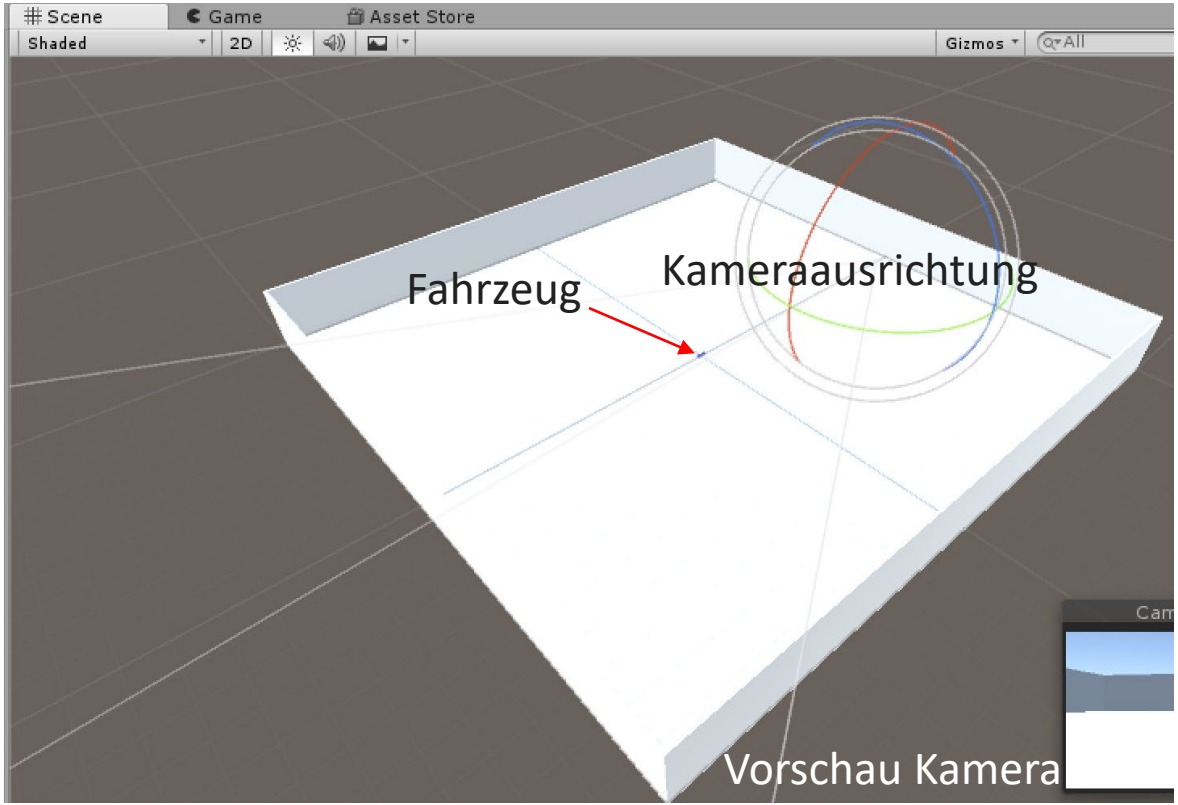


Das Spiel


File Edit Assets GameObject Component Window Help
Center Global
Collab Account Layers Layout
Hierarchy
Create All
unity_level-01
Main Camera
Directional Light
tracer-00
body
WheelFront
WheelBack
Level-01
BottomPlate
WallNorth
WallSouth
WallEast
WallWest
Scene Game Asset Store
Shaded 2D Gizmos All
Persp
Inspector
WheelBack Static
Tag Untagged Layer Default
Transform
Position X 0 Y 0.4 Z -1.1
Rotation X 0 Y 0 Z 90
Scale X 0.8 Y 0.5 Z 0.8
Cylinder (Mesh Filter)
Mesh Cylinder
Mesh Renderer
Lighting
Materials
Dynamic Occluded
TracerMaterial
Shader Standard
Add Component
Project Console
Create
Favorites
All Materials
All Models
All Prefabs
All Scripts
Assets
Scripts
Assets
Scripts
TracerMaterial
unity_level-01

Die Kamerasicht

Die Kamerasicht

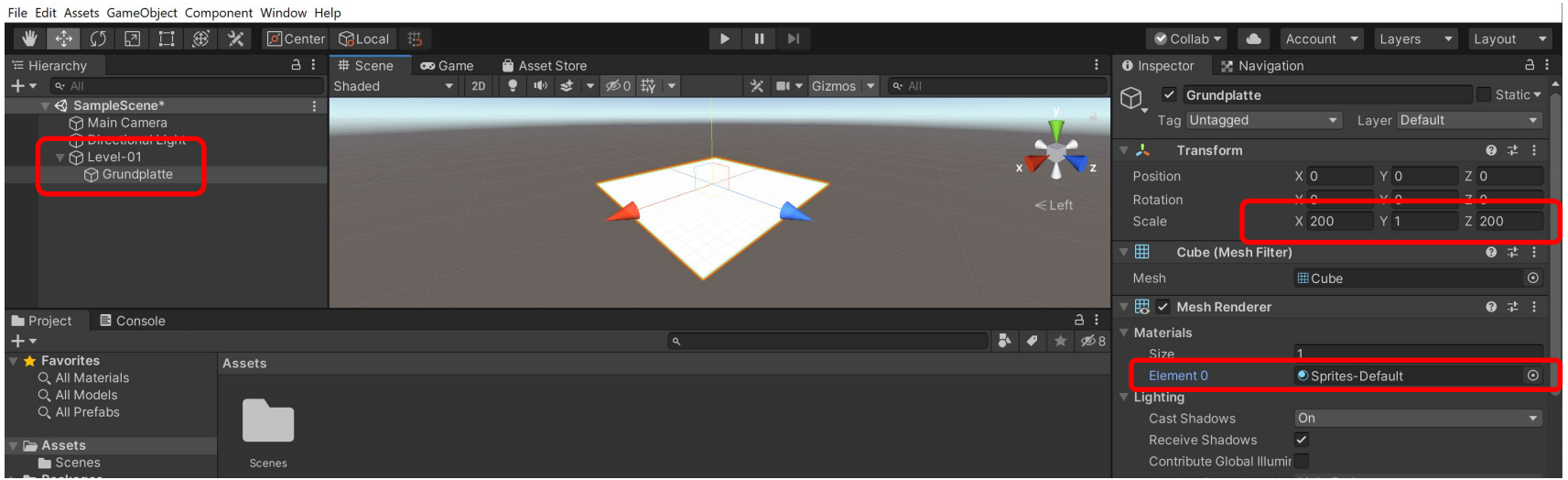


Die Kameraausrichtung kann durch das Einschalten der Rotations- oder Verschiebefunktion gesehen und geändert werden.

Erstellen der Umgebung für Level-01

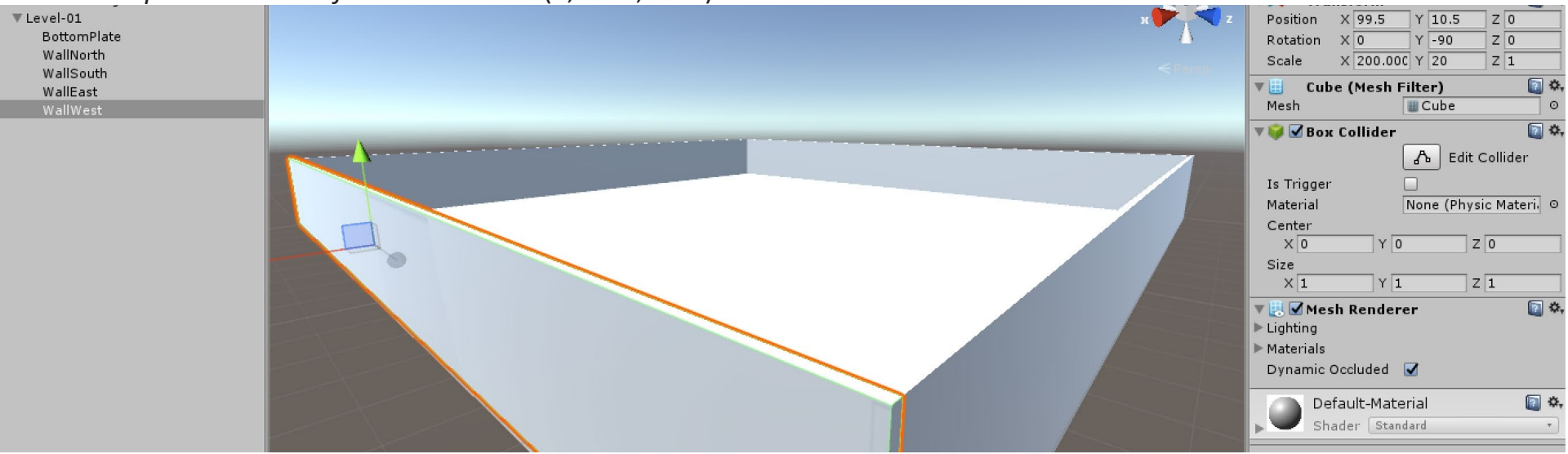
Wir erstellen ein leeres Spielobjekt, in dem wir die Umgebung erzeugen

1. *GameObject → Create Empty*
2. Umbenennen des Objekts in der Hierarchy durch Doppelklick auf den Namen und Überschreiben des Namens
3. *GameObject → 3D-Object → Cube* (Aus dem Cube formen wir eine Grundplatte)
4. *Inspector → Transform → Scale* (200, 1, 200) = (x, y, z)
5. *Inspector → Materials → Element0 → Sprites-Default*



Aus einem weiteren Würfel vier Seitenwände bauen

1. *GameObject* → *3D-Object* → *Cube* (Aus dem Cube formen wir eine Seitenwand.)
2. *Inspector* → *Transform* → *Scale* (200, 20, 1) = (x, y, z)
3. *Inspector* → *Transform* → *Position* (0, 10.5, -99.5) erstellen wir die Nordwand
4. Mit einem Klick mit der rechten Maus auf die Nordwand in der *Hierarchy* öffnen wir ein Menü → Mit *Duplicate* kopieren wir eine weitere Wand, die nun an die Südseite verschoben wird.
   Inspector → *Transform* → *Position* (0, 10.5, 99.5)

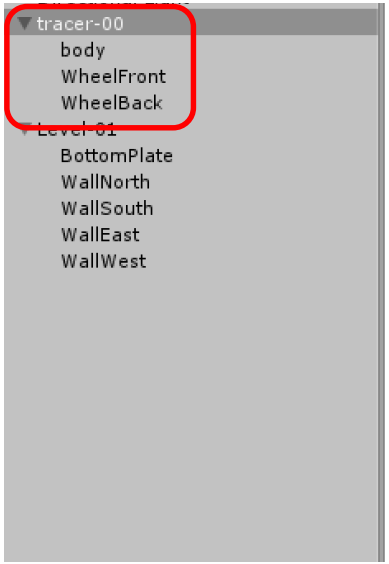


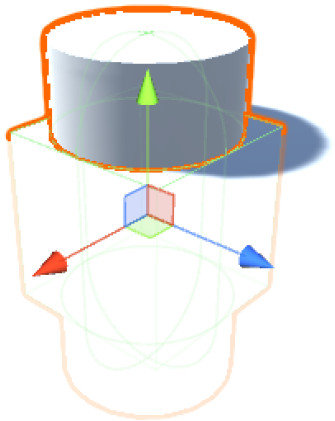
Duplizieren und Verschieben von Komponenten

1. Mit einem Klick mit der rechten Maus auf die Südwand in der *Hierarchy* öffnen wir ein Menü → Mit *Duplicate* kopieren wir eine weitere Wand, die nun an die Ostseite verschoben wird.
2. Position (-99.5, 10.5, 0), Rotation (0, 90, 0)
3. Wir duplizieren die Ostwand, um die Westwand zu erzeugen und mit den Koordinaten (99.5, 10.5, 0) wandert sie an die richtige Position.

4. Ein einfaches Fahrzeug modellieren

1. *GameObject → CreateEmpty*
2. Umbenennen des Objekts in Auto durch Doppelklick auf den Namen und Überschreiben des Namens.
3. *GameObject → 3D-Object → Cube* (umbenennen in Body)
4. *GameObject → 3D-Object → Cylinder* (Zyl. duplizieren und beide umbenennen in WheelFront und WheelBack)

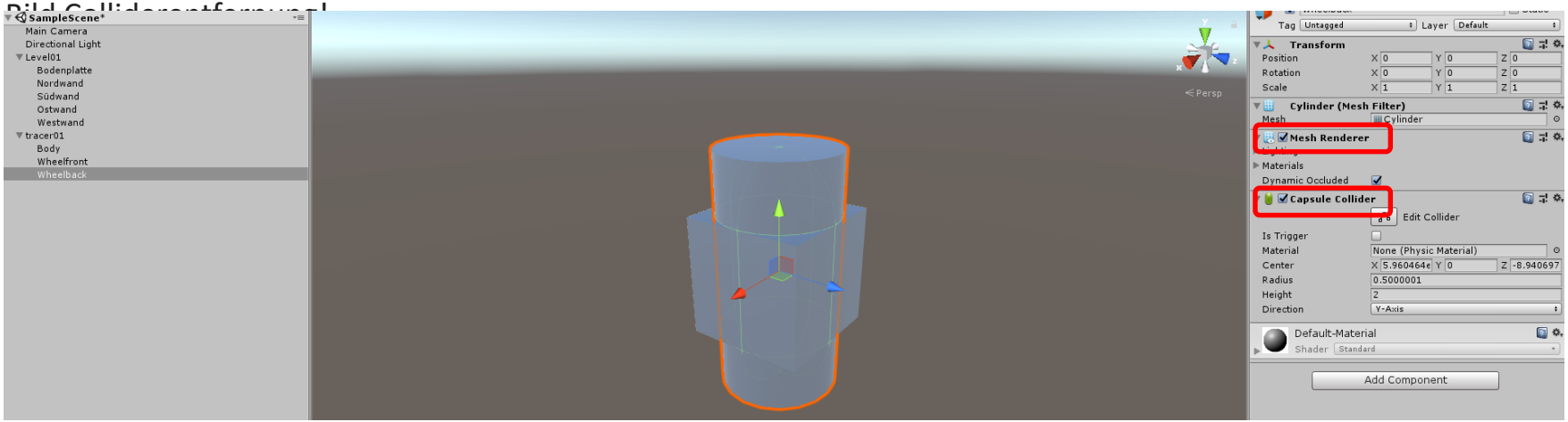




Die Collider entfernen

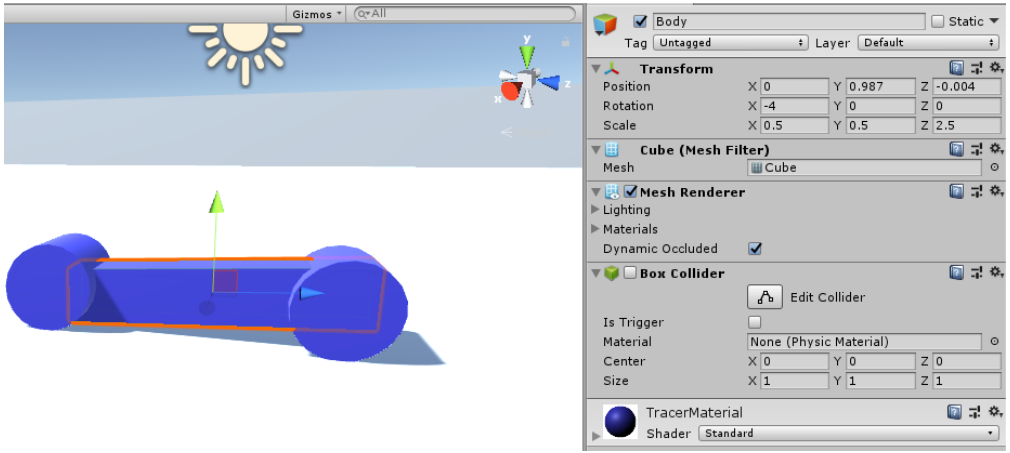
Wir benötigen später für alle drei Komponenten nur einen *Collider.* Deshalb deaktivieren wir zunächst die *Collider* der Komponenten Body, WheelFront und WheelBack.

Bild Colliderentfernung!

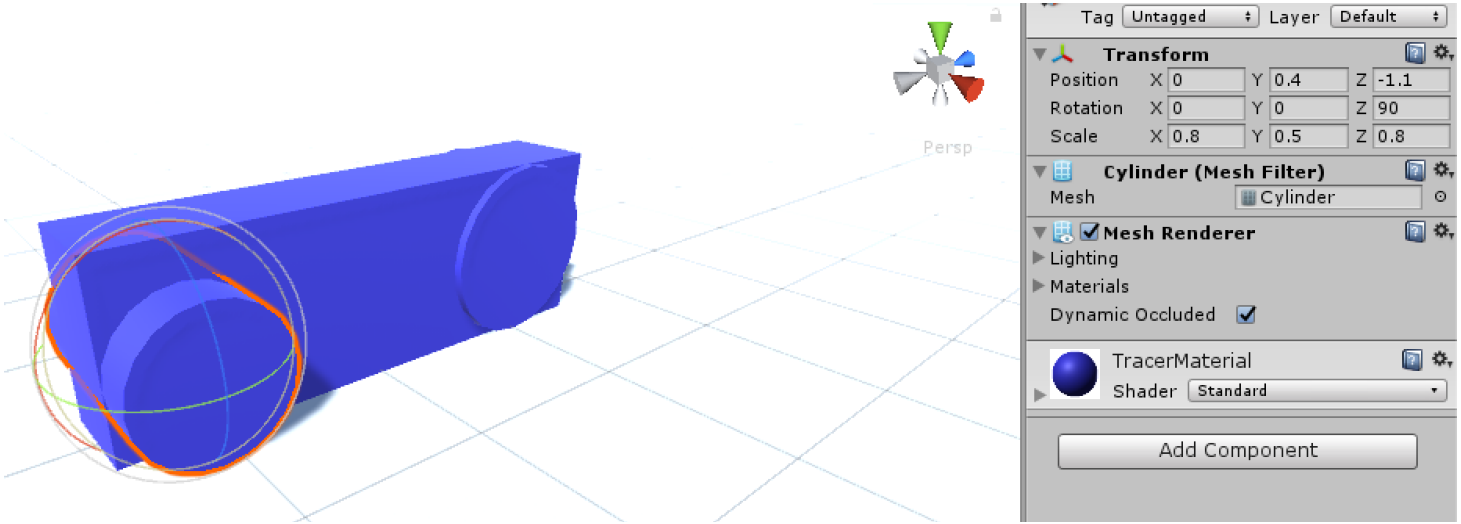


Tracer einfärben

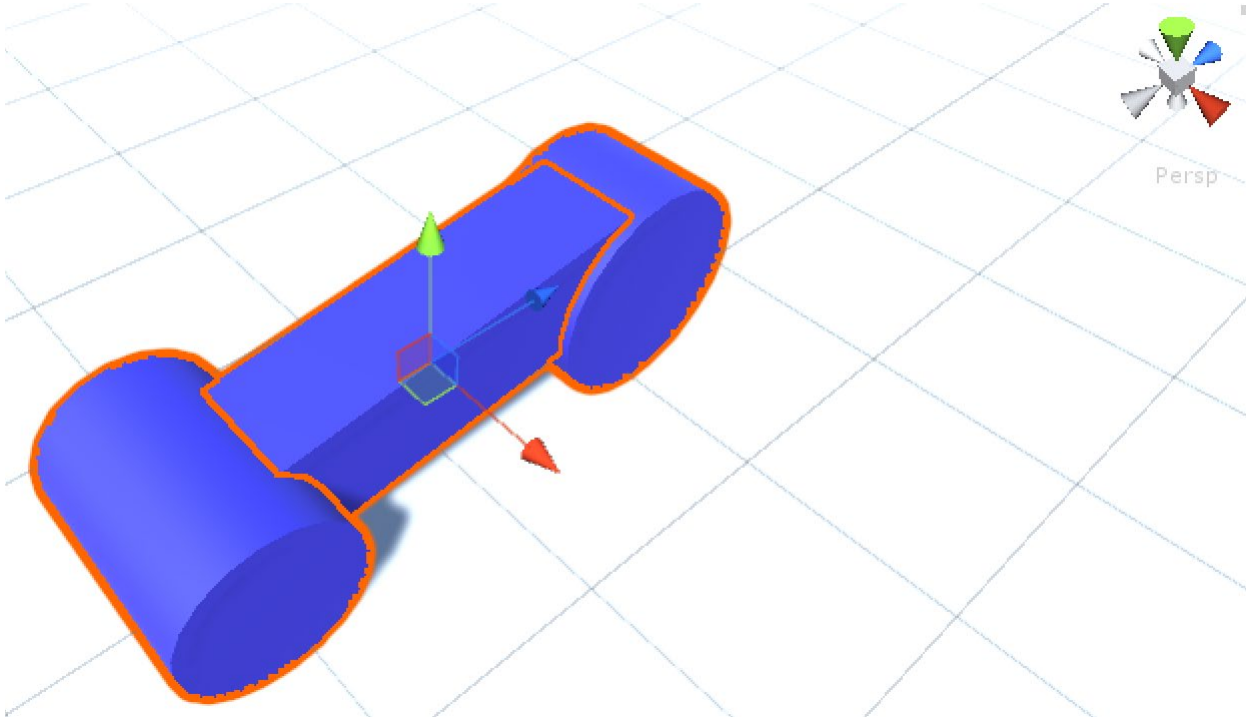
1. *Assets → Create → Material*
2. Einen Materialnamen vergeben z. B. TracerMaterial
3. Das Material im *Inspector* → Materials → Element 0 auf die drei Komponenten verteilen
4. Die Komponenten über der Bodenplatte positionieren, falls nicht schon geschehen
5. Den Body auf *Scale* = (0.5, 1, 3) setzen
6. Die Räder anpassen
7. Die Räder in Größe und Position einstellen



Räder gestalten



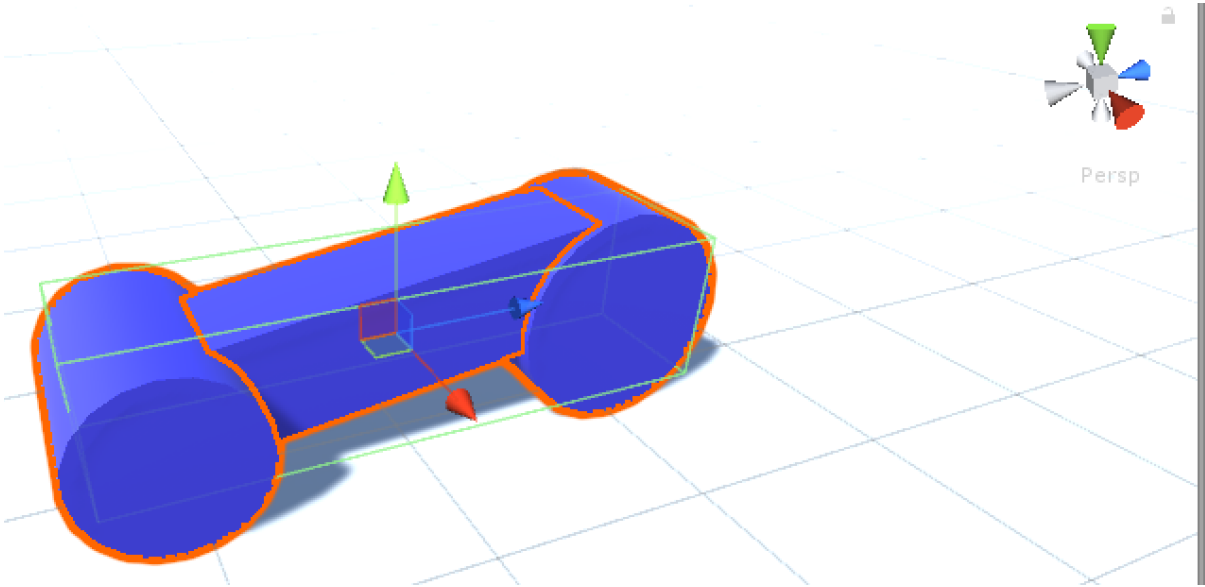
Rumpf gestalten

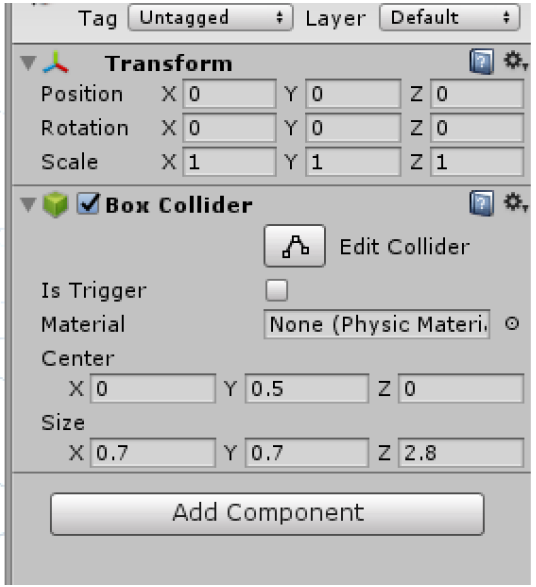


Einen Box Collider für das Gefährt erstellen

Zuerst müssen *alle Collider* der Einzelkomponenten entfernt sein! Danach kann der *Box Collider* für das Fahrzeug erstellt werden.

1. *Component* → *Physics* → *Box Collider*
2. *Box Collider* → *Center* → (0, 0, 0)
3. *Box Collider* → *Size* → (2, 1, 3.8)





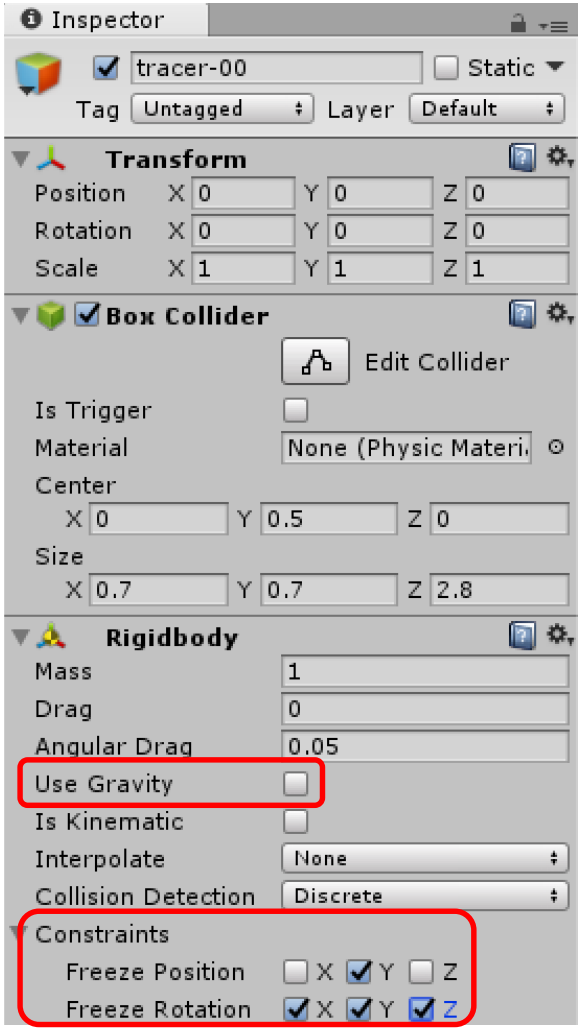
Fahrzeug bewegen und steuern

1. Die Physik-Engine für den Tracer aktivieren mit Rigidbody:
   Component → Physics → Rigidbody
2. *Use Gravity* deaktivieren
3. Bei den Constraints die vorgegebenen Einstellungen bei *Freeze Position* und *Freeze Rotation* vornehmen.

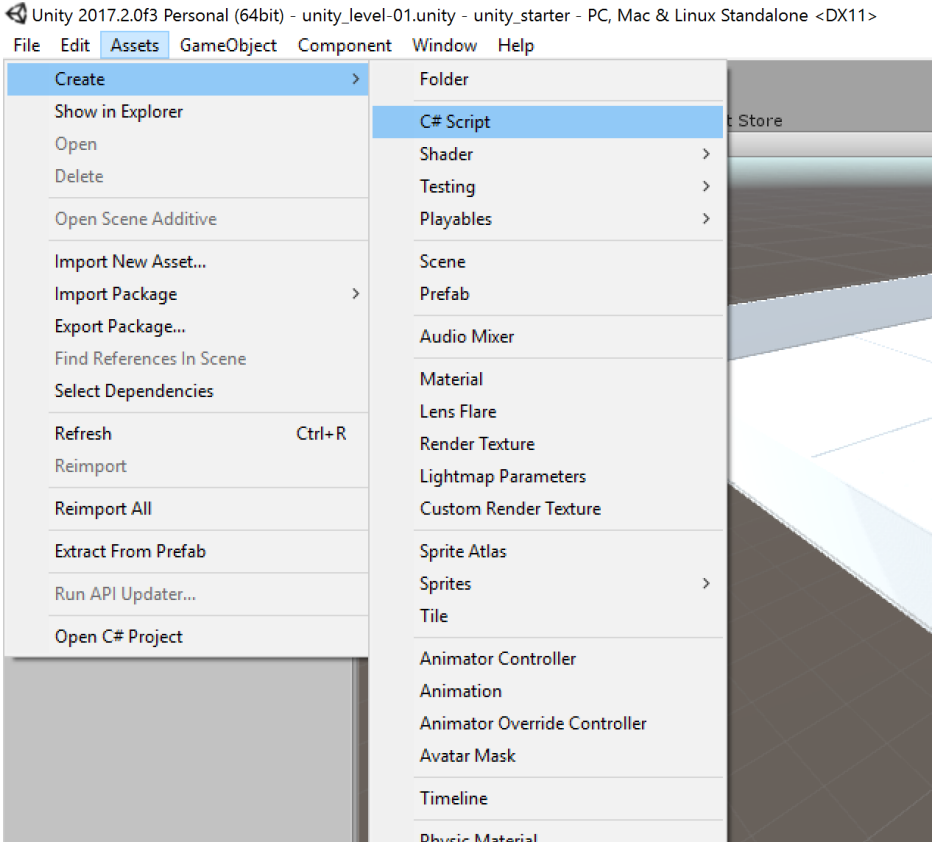
Wir benötigen in diesem Spiel keine Gravitation.

Die Rotation der Einzelteile stellen wir ab, da wir die Räder oder andere Bauteile nicht physikabhängig drehen lassen wollen.

Bei *„Freeze Position“* stellen wir nur die Bewegung in Y-Richtung (Höhe) ab. Da das Fahrzeug sich in der Ebene bewegen soll, müssen X- und Z-Richtung aktiv bleiben.



Fahrzeug mit C#-Programm in Bewegung setzen



Unter ***Assets → Create → C# Script*** wird ein Editor mit einem rudimentären Script geöffnet, das wir für unsere Zwecke abändern oder erweitern können.

Nennen Sie das Script in TracerController um!

Mit einem Doppelklick auf das *Script* unter den *Assets* öffnen Sie den Editor.

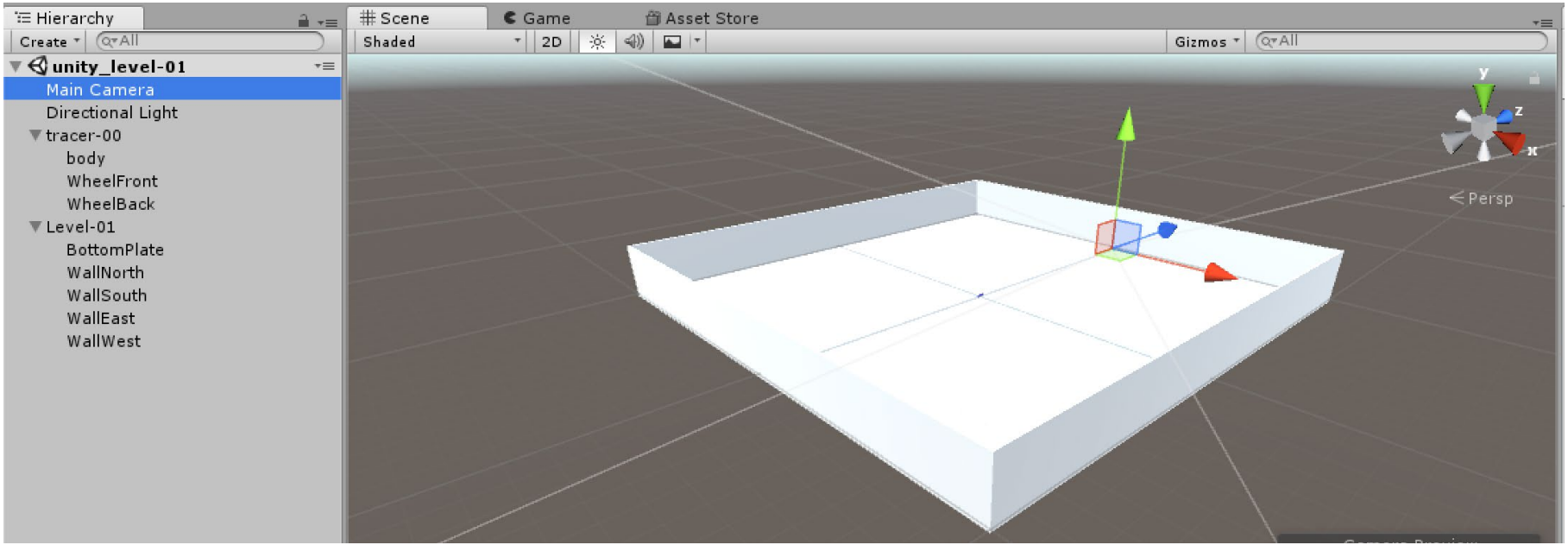
„Ein **Script** ist eine selbst geschriebene Komponente, die man wie jede andere Komponente an GameObjects in der Szene hängen kann und die für das GameObject ein bestimmtes Verhalten definiert.“ (2015 Chittesh)

6. Das erste C#-Script

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

    public class TracerController : MonoBehaviour {
        public float baseVelocity = 5F;        // 5 = Geschwindigkeit und F = Float
        private Rigidbody myRigidbody = null; // Variable, um auf die Physik des
                                              // Fahrzeugs zuzugreifen

        // Die folgenden Anweisungen werden nur beim Start ausgeführt
        void Start () {
            myRigidbody = GetComponent<Rigidbody>();
            myRigidbody.velocity = transform.forward * baseVelocity;

        }

        // Diese Anweisungen werden bei jedem Frame ausgeführt
        void Update () {

        }
    }
```

Erläuterung zu public und private → https://docs.microsoft.com/de-de/dotnet/csharp/language-reference/keywords/private

Kamera ausrichten



Fahrzeug mit Pfeiltasten steuern

```
void Update () {   // Die Methode Update wird 1 mal pro Frame aufgerufen!
        float rotation = 0F;  // Variable Rotation

        if (Input.GetKeyDown(KeyCode.LeftArrow)) {
            rotation = -90F;
        }
        else if (Input.GetKeyDown(KeyCode.RightArrow)) {
            rotation = 90F;
        }

        if (rotation != 0F) {
            transform.Rotate(0F, rotation, 0F); // Rotation um Y-Koordinate
            myRigidbody.velocity = transform.forward * baseVelocity;
        }
}
```

Im letzten Schritt muss das C#-Programm aus dem Asset auf das Objekt (Fahrzeug) in der Hierarchy gezogen werden, das sich bewegen soll.

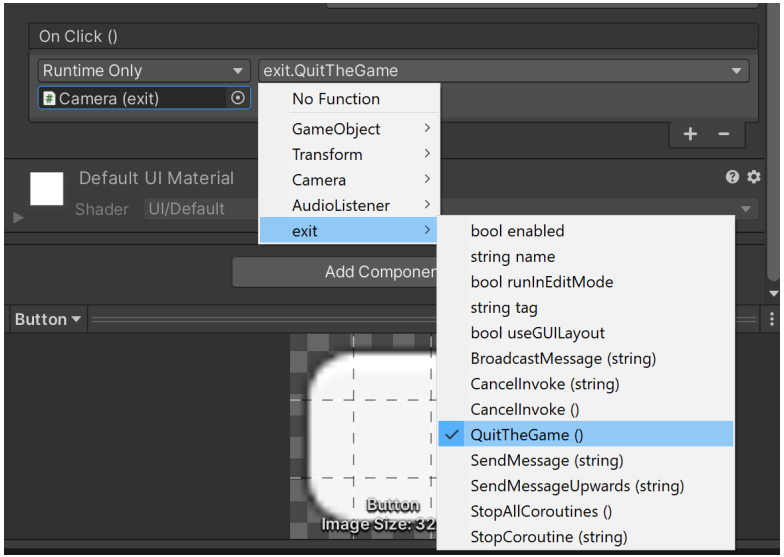
Einbau eines Exit-Buttons

1. Im Fenster Hierarchy mit der rechten Maustaste auf den Canvas-Eintrag klicken.
   Danach auf UI und dann auf Button klicken.
2. Die Größe des Buttons einstellen, damit er sichtbar und der Button-Text lesbar ist.
3. Den Button auf der Bildschirmoberfläche ausrichten über die "Anchor Presets".
   Das C#-Programm zum Ausstieg aus dem Programm mit Assets → Create → C# script erzeugen und den folgenden Programmcode einfügen.

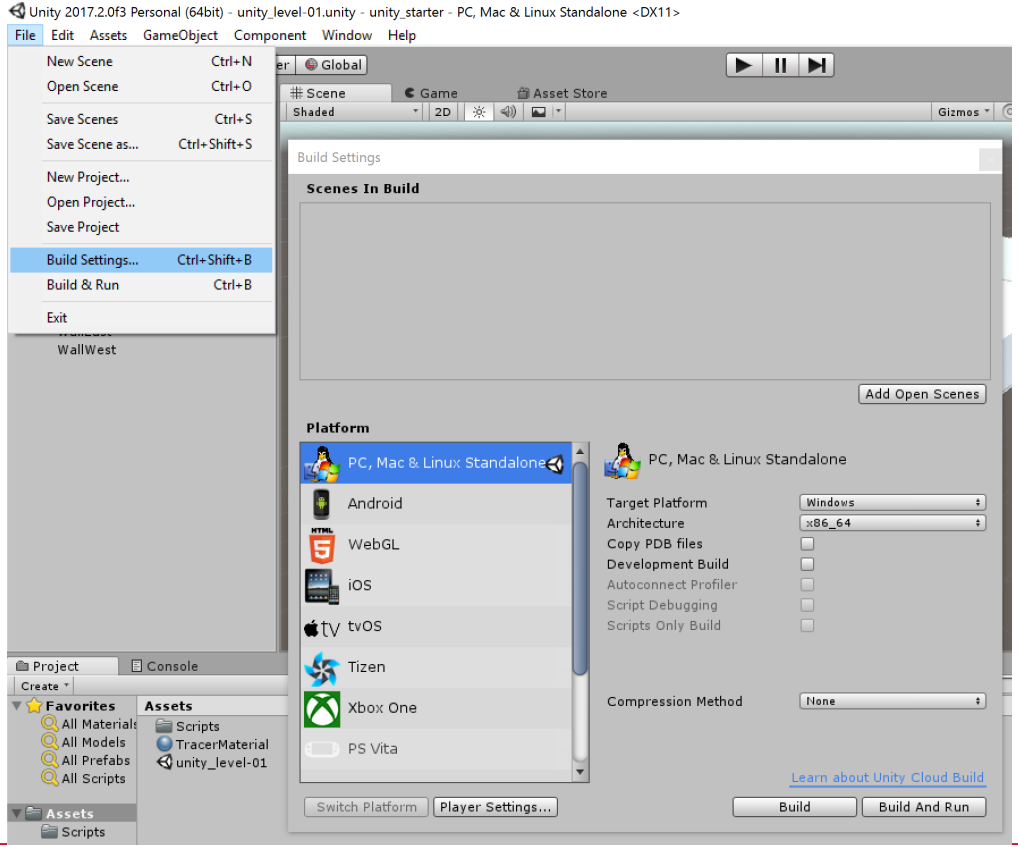
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class exit : MonoBehaviour
{
    public void QuitTheGame()
    {
        Application.Quit();
    }
}
```

Das Exit-Programm richtig zuweisen!

1. Ziehen Sie das Exit-Programm aus dem Assets-Fenster auf die Kamera.
2. Aktivieren Sie den Button und klicken sie im Fenster Inspector auf das „+“ unter On Click ().
3. Klicken Sie auf den kleinen runden Kreis mit der Bezeichnung "None (Object)".
4. Wählen Sie in dem erscheinenden Fenster den Reiter „Scene“.
5. Wählen sie in der Liste „Camera“ aus
6. Klicken sie auf das Feld rechts von „Runtime Only“.
7. Fahren Sie mit der Maus auf das richtige Programm und wählen sie im aufgehenden Fenster die gewünschte Funktion aus.
8. Starten sie "Build And Run" und testen sie die Funktion.



Das Spiel mit Build als Anwendung erstellen



Mit *File → Build Settings* öffnen wir das Fenster, um einzustellen, für welches Gerät oder welches Betriebssystem wir die Anwendung erstellen wollen.

Mit *Build* wird die Anwendung erstellt.

Literatur

Chittesh J. (2015) Das Unity-Buch 2D- und 3D-Spiele entwickeln. Heidelberg, Dpunkt-Verlag.

Seifert C. (2014) Spiele entwickeln mit Unity-Buch. München, Hanser-Verlag.

Kontakt:

Prof. Martin Schober | Informations- und Medientechnik

Hochschule Karlsruhe - Technik und Wirtschaft
Fakultät für Informationsmanagement und Medien

Postanschrift: Postfach 24 40, 76012 Karlsruhe
Besucheranschrift: Amalienstr. 81-87 | 76133 Karlsruhe | Raum AM 113
fon +49 (0)721 925 - 2990 | fax +49 (0)721 925 -1125
martin.schober@hs-karlsruhe.de | www.technischeredaktion.com/