# Dokumentation Workshop ITCS

# 01. Juli 2020

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Version** | **Datum** | **Autor** | **Status** | **Kommentar** |
| 2.1 | 24. Juni 2020 | Haimerl, Julian | fertig |  |
| 2.2 | 24. Juni 2020 | Kroll, Alexander | fertig | Textanpassungen |

## Zweck dieses Dokuments

Dieses Dokument dient der Vorbereitung des Messe Workshops ITCS.

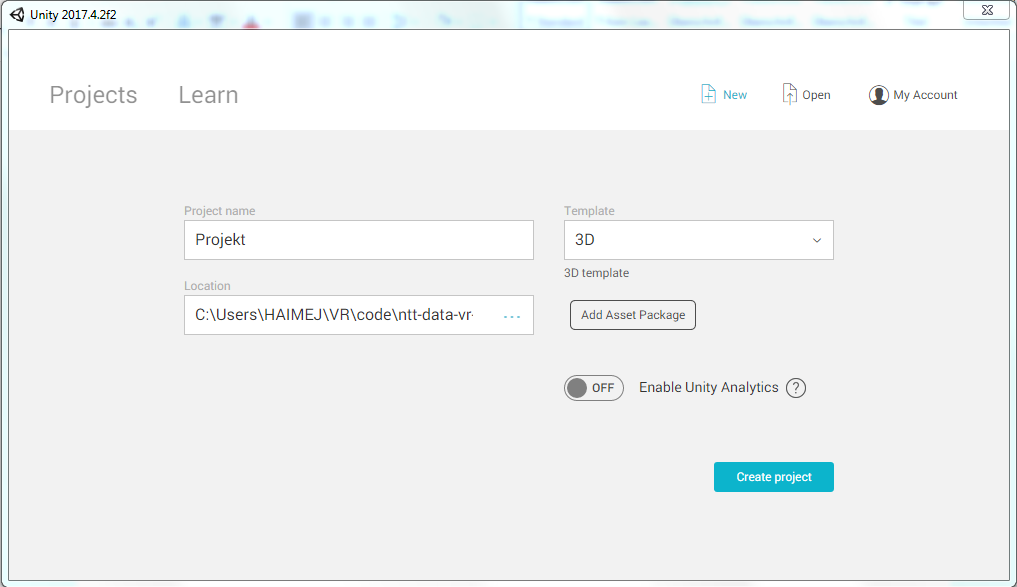
## Installation von Unity

Unity ist eine Entwicklungsumgebung, die in erster Linie zur Implementierung von Spielen genutzt wird. Es werden mächtige Tools bereitgestellt, mit welchen man bereits nach kurzer Zeit viele Konfigurationen am eigenen Projekt vornehmen kann.

Es Empfiehlt sich immer die neueste Version von Unity zu installieren. Den Download findet man online unter: <https://unity3d.com/de/get-unity/download>

Sollte man aber ein bereits bestehendes Softwareprojekt importieren wollen, sollte die gleiche Version genutzt werden, die zur Erstellung des Projekts genutzt wurde. Andernfalls können Inkompatibilitäts-Exceptions auftreten.

## Erstellen eines neuen Projekts

Beim Erstellen des Projekts muss bereits angegeben werden ob es sich um eine 2D oder 3D Anwendung handelt. Außerdem wird ein Name vergeben.

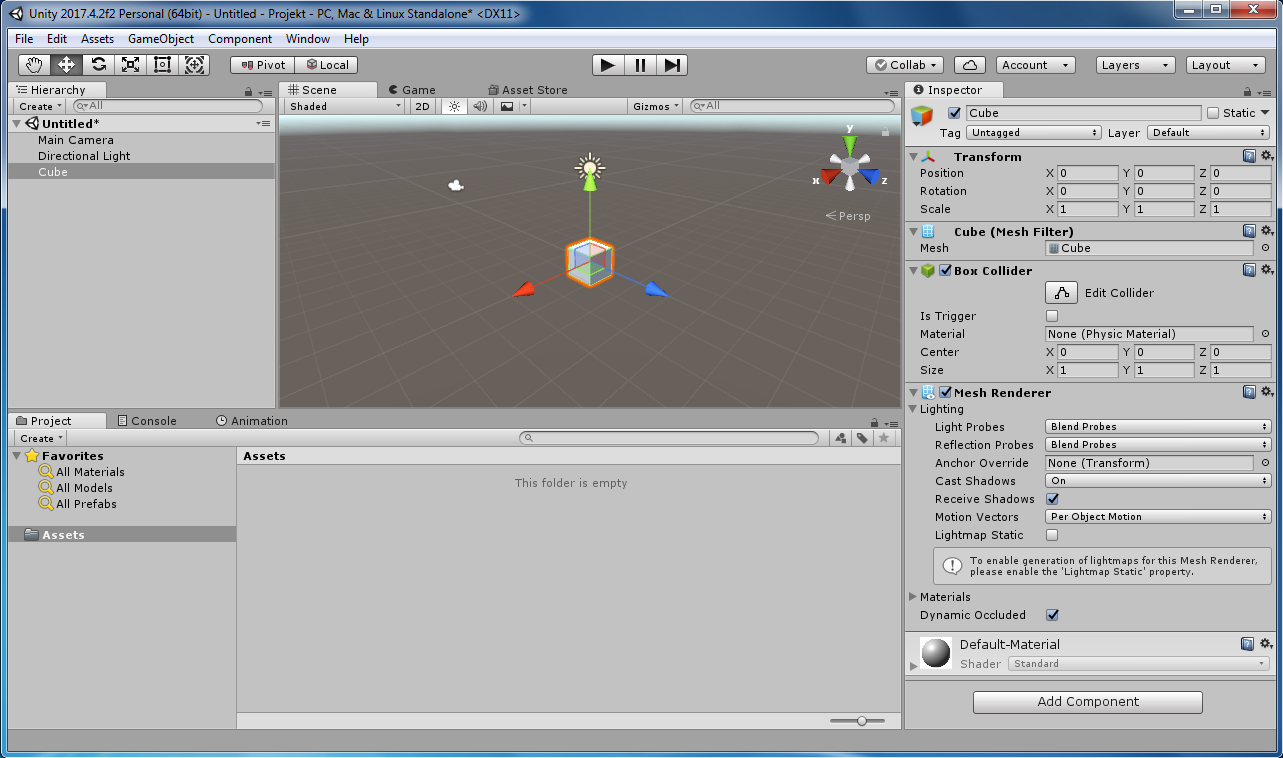
Im Anschluss erreicht man die Entwicklungsumgebung.

An dieser Stelle ist zu erwähnen das in Unity alle in der Anwendung enthaltenen Objekte als Objects bezeichnet werden. Jedes Object hat keine, eine oder mehrere Components welche die Eigenschaften/Komponenten eines Objects verkörpern.

So wäre ein Object beispielsweise ein Würfel welchen man über den Reiter GameObject -> 3D Object -> Cube einfügen kann.

Eine Component zu diesem Cube kann über die „Inspector“ Sicht hinzugefügt werden nachdem der Cube in der Hauptansicht, dem „Scene View“ ausgewählt wurde. In unserem Beispiel wäre die importierbare Component „Rigidbody“ denkbar, welche das Objekt von den physikalischen Gesetzten beeinflussbar macht, so dass man den Würfel also Bewegen/Fallen lassen/Weg Drücken kann.

Zu sehen auf dem unteren Screenshot.



Außerdem ist standartmäßig eine Lichtquelle und die Main Camera, welche die Sicht des Nutzers verkörpert, gegeben.

Die Anwendung kann über den „Play“ Knopf oben mittig gestartet werden.

## Anwendungsfallerstellung

## Terrain bereitstellen

Um nun eine ansehnliche Anwendung zu erstellen müssen wir zunächst den Grundstein dafür legen.  
Dieser soll ein „Terrain“ sein, welches den Untergrund bzw. Boden unserer Welt darstellt.

Nun erstellen wir also unsre Welt über GameObject -> 3D Object -> Terrain.

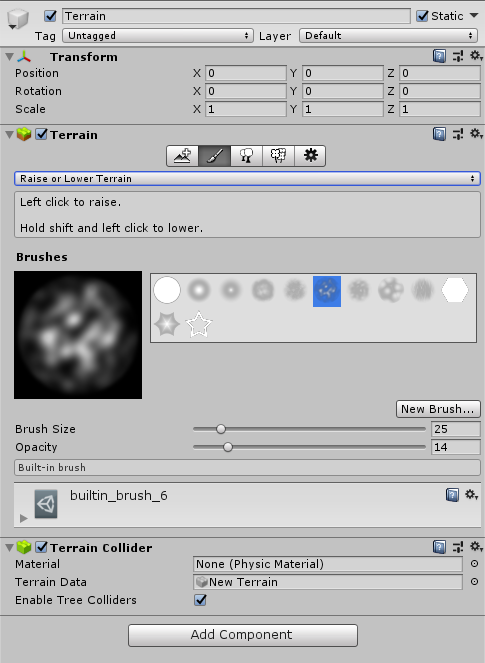
Schauen wir uns nun über den Inspektor die existierenden Components unserer Welt an.

Besonders wichtig ist die auf dem Screenshot Rot hervorgehobene Menüleiste über die wir das Terrain „verzerren“ können um so beispielsweise Berge, Täler oder andere 3D Landschaftsmerkmale einzufügen. Des Weiteren können weitere 3D Objekte wie Bäume oder andere Details auf das Terrain „gepflanzt“ werden. Die Größe und andere Globale Details der Welt können unter „Terrain Settings“ verändert werden.

Zunächst sollte die Größe des Terrains angepasst werden. Über die Terrain Settings (Zahnrad) lassen sich die Werte für „Terrain Width“ und „Terrain Length“ verändern. In beiden Fällen ist 250 genug für unsere Anforderungen.

Versuchen sie sich an den Verzerrungsfunktionen und erstellen sie eigene Hügel am Rande.

Verändern Sie hierfür die im Menüpunkt „Paint Terrain“🡪„Raise or Lower Terrain“ die Werte für „Brushsize“ und „Opacity“ um Breite und Höhe der Berge festzulegen und malen sie dann mit der Maus direkt auf ihr Terrain im Scene View.



Sind Sie mit Ihrer Bearbeitung zufrieden speichern Sie ihre Scene mit einem prägnanten Namen.

## Texturen einführen

Texturen verändern das Aussehen unseres Terrains, beispielsweise soll unsere Welt grundsätzlich mit Gras bedeckt sein aber die Berge sollen eine Felsoberfläche anstelle des Grases haben.

Dafür verwenden wir die mitgelieferten Texture Files. Diese sind in dem „textures“ Ordner, der über die Project View erreichbar ist, abgelegt.

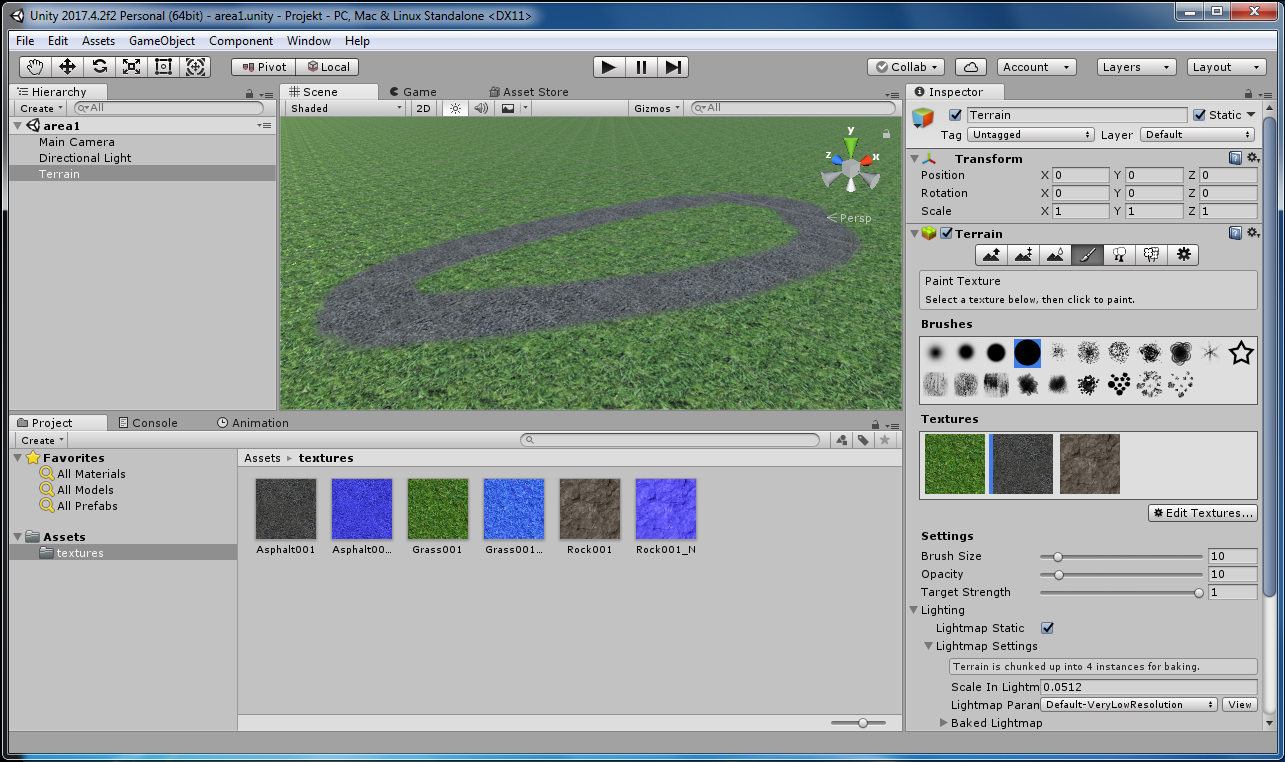
Fügen Sie die drei Texturen Grass001, Asphalt001 und Rock001 nach Auswahl des Terrains im Scene View über den Menüpunkt „Paint Terrain“ 🡪 „Paint Texture“ 🡪 „Edit Terrain Layers“ 🡪 „Create Layer“ einzeln nacheinander im Inspector View hinzu.

Um das ganze Terrain mit Gras zu bedecken sollte man nun im Inspector View die Gras Textur für das Terrain auswählen und daraufhin die die „Brush Size“ und die „Opacity“ auf den größten Wert verändern, damit man so viel Möglich auf einmal bemalen kann. Jetzt bemalt man das Terrain in dem Scene View vollständig.

Wiederholen Sie nun den Schritt für den Asphalt001.jpg und den Rock001.jpg File.  
Diese sollen wir nun auf die Graslandschaft aufmalen, hierbei ist Feingefühl gefragt.

Wählen Sie die Asphalt Textur in Ihrem Terrain Inspector aus. Verändern Sie die „Brush Size“ und die „Opacity“ jeweils auf den Wert 20 und malen Sie eine Straße auf Ihre „Graswelt“.

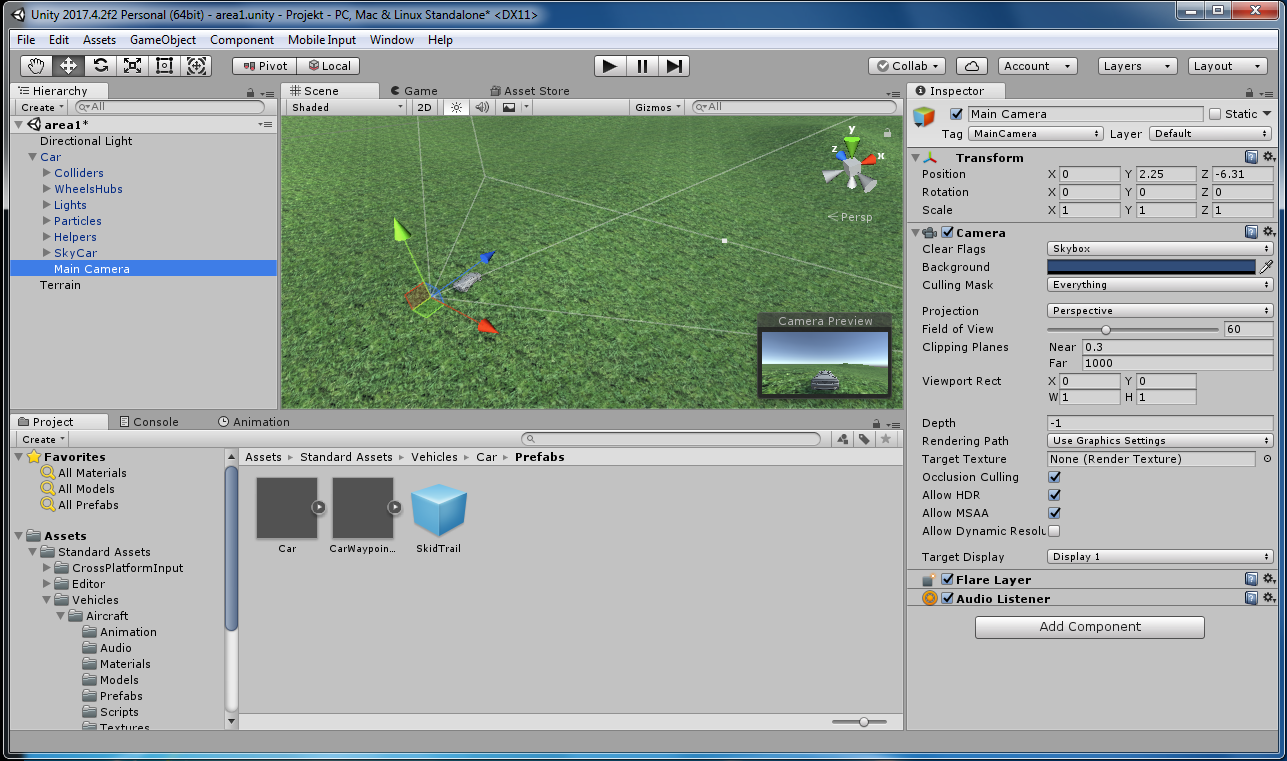
An diesem Punkt sind sie bereits in der Lage Ihre Welt auf enorme weise zu individualisieren.  
Versuchen sie sich auch daran Berge mit der „Rock“ Textur zu gestalten.



## Fahrzeug einführen

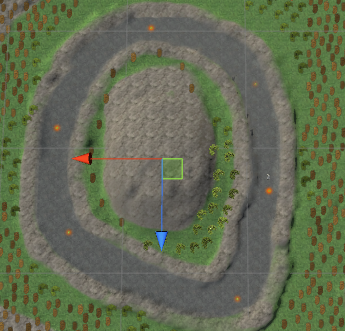
Das Template enthält schon alle benötigten Daten, weshalb wir diese nicht importieren müssen. Im Ordner Standard Standard Assets finden sie die Objekte „CarWaypointBased“ und „Car“ welche Sie wiederum per Drag & Drop in ihre Scene ziehen können. Es handelt sich hierbei um fertige Objekte deren Zusammensetzung nachfolgend erklärt wird.

Zuerst braucht das Auto die Main Camera als Komponente. Dies erreichen wir, indem wir die „Main Camera“ im Hierarchy View per Drag & Drop in das „Car“ Object hineinziehen.   
Nun müssen wir noch die Position der Kamera ändern. Klicken Sie hierfür auf das Main Camera Object und ändern Sie im Inspector View die X, Y und Z Werte der Position & Rotation auf 0.   
Um nun die Optimale Camera Position zu finden sollten wir das „Move Tool“ ganz links Oben (siehe Screenshot) auswählen um die Y und Z Position des Autos so zu verschieben wie sie am angenehmsten erscheint. Beispielsweise leicht erhöht hinter dem Auto oder als First Person Perspektive. Beachten Sie hierfür den „Camera Preview“ im Scene View.



Probieren Sie Ihre Anwendung nun aus. Haben Sie das „CarWaypointBased“ gewählt sollte das Fahrzeug nun willkürlich über ihre Straße steuern. Das „Car“ hingegen können Sie nun mit den Tasten „W A S D“ über die Strecke steuern.

## Autonome Strecke erstellen (nur notwendig für „CarWaypointBased“)



Wie der Name unseres Fahrzeuges evtl. schon suggeriert ist die Idee hinter der autonomen Funktionalität das wir nacheinander verschiedene Wegpunkte abfahren.

Dafür erstellen wir 6 „Cube“ 3D Objekte die wir so auf der Strecke verteilen das Sie zueinander über lineare Wege erreichbar sind.

Außerdem sollten bei diesen Cubes die „Box Collider“ sowie die „Mesh Renderer“ Components deaktiviert werden damit diese nicht mehr sichtbar bzw. berührbar sind.

Außerdem brauchen wir einen weiteren Cube auf der Position unseres 1. Cubes, den wir Tracker nennen und etwas vergrößern. Dieser soll von Wegpunkt zu Wegpunkt springen und das dauerhafte Ziel unseres Fahrzeuges sein. Hier müssen wir nun auch den „Mesh Renderer“ deaktivieren damit wir den Tracker Cube nicht im finalen Simulator sehen können. Für Debugging zwecke wäre das hingegen gut geeignet.  
Darüber hinaus müssen wir unter dem „Box Collider“ den Haken bei „is Trigger“ setzen da wir folgendes Skript an unseren Würfel anhängen und durch Kollision mit dem Fahrzeug auslösen wollen:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class AutoRoute : MonoBehaviour {

public GameObject InitialPoint;

public GameObject Point01;

public GameObject Point02;

public GameObject Point03;

public GameObject Point04;

public GameObject Point05;

public GameObject Point06;

public int MarkTracker;

// Update is called once per frame

void Update () { // jeden frame wird das ziel je nach position neu berechnet

if (MarkTracker == 0) {

InitialPoint.transform.position = Point01.transform.position;

}

if (MarkTracker == 1) {

InitialPoint.transform.position = Point02.transform.position;

}

if (MarkTracker == 2) {

InitialPoint.transform.position = Point03.transform.position;

}

if (MarkTracker == 3) {

InitialPoint.transform.position = Point04.transform.position;

}

if (MarkTracker == 4) {

InitialPoint.transform.position = Point05.transform.position;

}

if (MarkTracker == 5) {

InitialPoint.transform.position = Point06.transform.position;

}

}

IEnumerator OnTriggerEnter(Collider collision) {

if (collision.gameObject.tag == "Dreamcar01") { // Tag unseres Autos

this.GetComponent<BoxCollider> ().enabled = false; // Objekt deaktivieren

MarkTracker += 1;

if (MarkTracker == 6) {

MarkTracker = 0; // wieder von vorne Anfangen

}

yield return new WaitForSeconds (1); //warten weil sonst die lenkung des Autos sehr abrupt ist

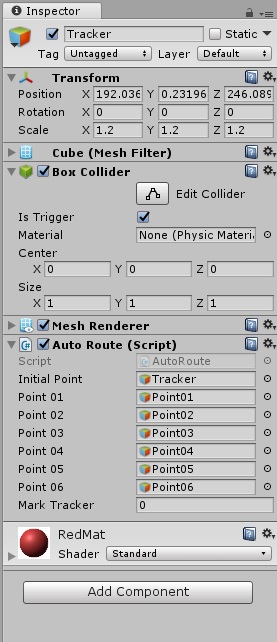
this.GetComponent<BoxCollider> ().enabled = true; // Objekt aktivieren

}

}

}

Wenn Sie das C# Skript erstellt und Ihrem Tracker Objekt zugeordnet haben, müssen noch die Variablen initialisiert werden, ziehen Sie hierfür Ihre Wegpunkte sowie das Tracker Objekt selbst in die dafür vorgesehenen Felder des Tracker Objekts:

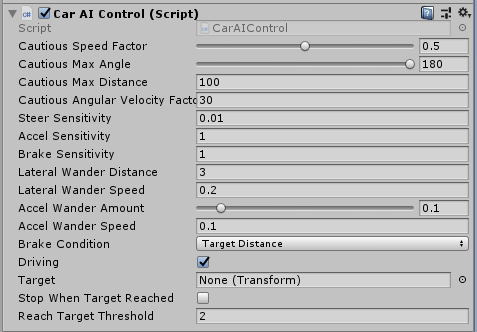


Außerdem müssen wir noch den Tag „Dreamcar01“ erstellen („Add Tag“ unter Tag im Inspector) und allen Collider Objekten des Fahrzeugs zuordnen um den Trigger im Script auslösen zu können.



Damit das „autonome Fahrzeug“ unserem Tracker Objekt folgt, müssen wir dem Auto noch sagen das es eben diesem Objekt hinterherfahren muss.

Wir werfen einen Blick auf das Fahrzeug Objekt und schauen und im Inspector View das Skript „Car AI Control“ genauer an. Die Variable „Target“ gibt dem autonomen Fahrzeug ein Ziel für die aktuelle Fahrtrichtung an. Wenn sich dieser Punkt in der Welt bewegt wird das Fahrzeug ihm folgen.



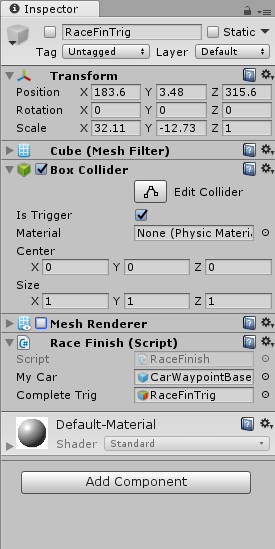
## Fahrt nach Zwei Runden automatisch beenden

Dafür müssen wir zunächst drei Cubes erstellen welche wir auf alle Fälle passieren, wenn wir über unsere Strecke fahren.

Vorschlag:



Einer davon in der Ziellinie (**FinPointTrigger**), den zweiten kurz hinter der Ziellinie (**RaceFinTrigger**) und den dritten auf halber Strecke (**HalfPointTrigger**). Dabei sollen der FinPointTrigger und der RaceFinTrigger deaktiviert werden. Außerdem müssen bei allen dreien die „Mesh Renderer“ deaktiviert und der Haken bei „is Trigger“ gesetzt werden.

Beispiel „RaceFinTrigger“:

Zuletzt müssen wir den drei Triggern noch 3 Skripte zuordnen welche die Funktionalität beinhalten sowie die Game Objekte initialisieren die wir im Code beeinflussen. Erklärt in den Kommentaren:

**HalfPointTrigger**:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class HalfLapTrigger : MonoBehaviour {

public GameObject LapCompleteTrig;

public GameObject HalfLapTrig;

void OnTriggerEnter ()

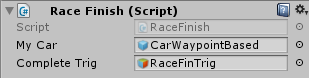
{

LapCompleteTrig.SetActive(true); // aktiviert Trigger für vollständige Runde

HalfLapTrig.SetActive(false); // deaktiviert sich selbst

}

}

**FinPointTrigger**:

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

public class LapCompleteTrigger : MonoBehaviour {

public GameObject LapCompleteTrig;

public GameObject HalfLapTrig;

public GameObject RaceFinTrig;

public int LapsDone;

void OnTriggerEnter()

{

LapsDone += 1; // Zählt bei passieren Rundenzah hoch

if (LapsDone == 2) {

RaceFinTrig.SetActive(true); //falls Runden gleich 2: Fahrt per Trigger beenden

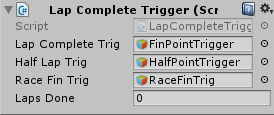
}

HalfLapTrig.SetActive(true); // aktiviert Trigger für halbe Runde

LapCompleteTrig.SetActive(false); // deaktiviert sich selbst

}

}

**RaceFinTrigger**:

using System;

using System.Collections;

using System.Collections.Generic;

using UnityEngine;

using UnityEngine.UI;

public class LapComplete : MonoBehaviour

{

public GameObject LapCompleteTrig;

public GameObject HalfLapTrig;

public int LapsDone;

public GameObject RaceFinish;

public GameObject LapCounter;

void OnTriggerEnter()

{

LapsDone += 1;

if (LapsDone == 2) {

RaceFinish.SetActive(true);

}

LapCounter.GetComponent<Text> ().text = "" + LapsDone;

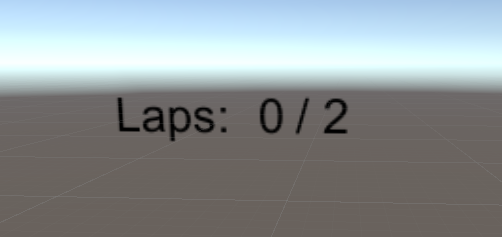
HalfLapTrig.SetActive(true);

LapCompleteTrig.SetActive(false);

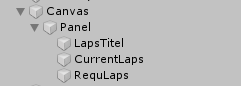
}

}

Wie man es dem Code entnehmen kann aktivieren sich die Trigger nacheinander gegenseitig bis 2 Runden gefahren sind. Danach wird der finale Trigger aktiviert, welcher die Fahrt dann endgültig beendet indem er alle relevanten Fahrzeugkomponenten deaktiviert.



Zuletzt soll ein Rundenzähler umgesetzt werden, der als UI Element für den User angezeigt werden soll. Dafür muss ein Canvas Game Objekt erstellt werden, unter dem wir dann 3 weitere Text UI Elemente anlegen. Das macht man über einen Rechtsklick 🡪 UI 🡪 Text.



Der Titel und die Maximale Rundenanzahl sind einfache Textelemente, die nicht verändert werden.  
Die aktuelle Rundenanzahl wird allerdings im „LapComplete” Skript angepasst. Damit man das Objekt erreichen kann muss man es als Variable im besagten Skript deklarieren. Auch hier erreichen wir das über Drag&Drop im Inspector View.