

Dokumentation Golfinity VR

Virtuelle Realität und Animation SS 23

Patrick Flögel	85824
Felix Kurz	86041
Markus Kohlstrunk	86765
Lukas Deli	86764

Inhaltsverzeichnis

Spielidee

Verwendete Assets

Skripte

Zusätzliche Abgabe: Blender Datei

Spielidee:

Als Spielidee haben wir uns Minigolf ausgesucht. Unser Spiel orientiert sich dabei stark an realitätsnahen Maßen und Regeln. Man spielt einen Rundkurs, welcher 18 Bahnen enthält und man eben mit möglichst wenig Schlägen durchkommen soll.

Dadurch, dass das Spiel eben nur ein Spiel ist, kann man sich einige Bonusfeatures zu Nutze machen. Hiermit ist gemeint, dass sobald der Ball im Gras landet er an den Anfang der Bahn zurückgesetzt wird.

Des Weiteren haben wir uns überlegt, dass jede Bahn ein kleines Holzschild besitzt, welches Schläge und Highscore anzeigt. Zuletzt gibt es noch ein großes Holzschild, welches die Gesamtschläge und den Gesamthighscore anzeigt. Sobald man bei einer Bahn eingelocht hat, werden der Ball und man selbst zur nächsten Bahn teleportiert.

Verwendete Assets:

Für unser Projekt haben wir nur wenige Assets verwendet. Alle Assets, die hier nicht aufgelistet werden, gehen aus Eigenproduktion hervor.

Gras Textur:

<https://www.istockphoto.com/de/foto/nahtlose-rasen-gras-wiederholen-rasen-textur-gm1159879162-317320586>

Wald Backdrop Textur:

Enthalten im Standardumfang des Computerspiels OMSI II

Baum 3D-Objekt:

<https://sketchfab.com/3d-models/low-poly-tree-70f0e767fc2f449fa6fef9c2308b395f>

Ambiente Sound Effekt:

<https://getsoundly.com>

Alle weiteren Sound Effekte:

<https://www.zapsplat.com>

Skripte:

1. Bahn.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class Bahn : MonoBehaviour
6  {
7
8      // Das Schild, das zu dieser Bahn gehört
9      public BahnSchild schild;
10
11     // Der Ursprung der Bahn
12     public Vector3 bahnOrigin;
13
14     public Vector3 playerSpawnPoint;
15
16
17
18
19     // Start is called before the first frame update
20     void Start()
21     {
22         // Fülle globale Variablen
23         bahnOrigin = transform.position + new Vector3(0, 0.2f, 0.3f);
24         playerSpawnPoint = transform.position - new Vector3(0, 0, 0.3f);
25         //schild = transform.GetChild(1).gameObject.GetComponent<BahnSchild>();
26         int count = 0;
27         while (count < transform.childCount) {
28             if (transform.GetChild(count).gameObject.GetComponent<BahnSchild>() != null) {
29                 schild = transform.GetChild(count).gameObject.GetComponent<BahnSchild>();
30                 break;
31             }
32             count++;
33         }
34     }
35
36     // Update is called once per frame
37     void Update()
38     {
39
40     }
41 }
```

Dieses Skript ist als Component zu jeder Bahn hinzugefügt. Es stellt für das Script Schlagzähler.cs die Assoziation zu dem zur Bahn gehörenden Schild zur Verfügung, sowie die Positionen, an die der Ball und der Spieler teleportiert werden können. Das Schild wird automatisch aus den Kind-Objekten des Bahn-Objekts gesucht. Die Positionen ergeben sich dem Ursprungspunkt der Bahn, welcher immer an der Vorderkante der Bahn liegt.

2. Bahnschild.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using TMPro;
5
6  public class Bahnschild : MonoBehaviour
7  {
8
9      private int highscore = 0;
10     private int schlaege = 0;
11
12     private TextMeshPro bahnName;
13     private TextMeshPro schlaegeZaehler;
14     private TextMeshPro highscoreZaehler;
15
16     // Start is called before the first frame update
17     void Start()
18     {
19         PlayerPrefs.SetInt("Bahn 1", 5);
20
21         bahnName = transform.GetChild(0).gameObject.GetComponentInChildren<TextMeshPro>();
22         schlaegeZaehler = transform.GetChild(1).gameObject.GetComponentInChildren<TextMeshPro>();
23         highscoreZaehler = transform.GetChild(2).gameObject.GetComponentInChildren<TextMeshPro>();
24
25         setSchlaegeText();
26         loadHighscore();
27     }
28
29     // Update is called once per frame
30     void Update()
31     {
32
33     }
34
35     public void updateHighscore() {
36         if ((schlaege < highscore || highscore == 0) && schlaege > 0) {
37             highscore = schlaege;
38             setHighscoreText();
39             storeHighscore();
40         }
41     }
42
43     private void storeHighscore() {
44         PlayerPrefs.SetInt(bahnName.text, highscore);
45     }
46
47     private void loadHighscore() {
48         highscore = PlayerPrefs.GetInt(bahnName.text);
49         setHighscoreText();
50     }
51 }
```

Dieses Skript ist als Component zu jedem kleinen Schild hinzugefügt, das am Rand einer Bahn steht. Es verwaltet den auf dem Schild stehenden Text sowie die Anzahl der Schläge auf der Bahn und den Bahnhighscore. Die Text-Objekte werden automatisch aus den Kind-Objekten des Schild-Objekts gesucht. Über das „PlayerPrefs“-Objekt werden die Highscores auf der Festplatte gespeichert, damit sie auch nach dem Neustart des Programms noch zur Verfügung stehen.
(Fortsetzung des Scripts auf Folgeseite)

```
52     public void addSchlag() {
53         schlaege++;
54         setSchlaegeText();
55     }
56
57     public void resetSchlaege() {
58         schlaege = 0;
59         setSchlaegeText();
60     }
61
62     public int getSchlaege() {
63         return schlaege;
64     }
65
66     private void setSchlaegeText() {
67         schlaegeZaehler.text = "Schläge " + schlaege;
68     }
69
70     private void setHighscoreText() {
71         if (highscore == 0) {
72             highscoreZaehler.text = "Highscore none";
73         } else {
74             highscoreZaehler.text = "Highscore " + highscore;
75         }
76     }
77
78     public string getBahnName() {
79         return bahnName.text;
80     }
81
82 }
```

3. Schlagzaehler.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using TMPro;
4  using UnityEngine;
5  using System.Collections;
6  using System.Collections.Generic;
7  using System.Linq;
8  using UnityEngine.UI;
9  using Unity.XR.CoreUtils;
10 using UnityEngine.XR.Interaction.Toolkit;
11
12 public class Schlagzaehler : MonoBehaviour
13 {
14
15     // Sound, der abgespielt wird, wenn der Schläger den Ball berührt
16     [SerializeField]
17     public AudioSource schlagSound;
18
19     // Text, der die Gesamtanzahl der bisherigen Schläge auf dieser Bahn seit Start anzeigt
20     [SerializeField]
21     public TextMeshPro zaehlerText;
22
23     // Text, der den Highscore aller bisher beendeten Spiele auf diesem Parcours seit Installation anzeigt
24     [SerializeField]
25     public TextMeshPro highScoreText;
26
27     // Liste aller Bahnen des Parcours
28     [SerializeField]
29     public Bahn[] bahnen;
30
31     [SerializeField]
32     public XROrigin player;
33
34
35     // Zähler aller Schläge auf dem Parcours seit Start des Spiels
36     private int globalZaehler = 0;
37
38     // Highscore aller bisher beendeten Spiele auf diesem Parcours seit Installation des Spiels
39     private int globalHighscore = 0;
40
41
42     // Speicherung der aktuell zu bespielenden Bahn
43     private int currentBahn = 0;
44
45
46     // Speicherung des Schildes und der Startposition zu jeder Bahn
47     //#[SerializeField]
48     //private Dictionary<int, Bahn> bahnen = new Dictionary<int, Bahn>();
49
50
51     // Rigidbody des Balls
52     private Rigidbody rb;
53
54     bool hasCollided = false;
55
56     GameObject club;
57
58     Vector3 origin;
59
60     Color originalColor;
```

```

62     // Start is called before the first frame update
63     void Start()
64     {
65         // Befülle origins mit den jeweiligen Startpositionen des Balls.
66         // ...
67
68         // Lade den Highscore auf dieser Map
69         globalHighscore = PlayerPrefs.GetInt("GesamtHighscore");
70
71         // Setze die Texte auf dem Gesamt Schild
72         setZaehlerText();
73         setHighScoreText();
74
75         rb = GetComponent<Rigidbody>();
76
77         originalColor = GetComponent<Renderer>().material.color;
78
79         // Teleport Ball zur ersten Bahn
80         teleportBall();
81     }
82
83     // Update is called once per frame
84     void FixedUpdate()
85     {
86         //rb.velocity *= 0.9F;
87         if (rb.velocity.magnitude > 0) {
88             //Debug.Log("is Moving ...");
89         }
90
91         if (hasCollided && rb.velocity.magnitude < 0.05F)
92         {
93             club.GetComponent<Collider>().isTrigger = false;
94             GetComponent<Renderer>().material.color = originalColor;
95
96             if (bahnen[currentBahn].schild.getSchlaege() == 15) {
97                 currentBahn++;
98                 teleportBall();
99                 teleportPlayer();
100            }
101        }
102        /*if (Input.GetKeyDown(KeyCode.Space))
103        {
104            gameObject.transform.position = origin;
105            rb.velocity = Vector3.zero;
106            rb.angularVelocity = Vector3.zero;
107        }
108        */
109    }

```

```

115     private void OnCollisionEnter(Collision col) {
116
117         if (col.gameObject.CompareTag("Club")) {
118             // Spiele den Schlagsound ab
119             schlagSound.Play();
120             //xr.SendHapticImpulse(0.7f, 2f);
121
122
123             // Zähle einen neuen Schlag auf dem Schild mit der Gesamtübersicht
124             globalZaehler++;
125             setZaehlerText();
126
127             // Zähle einen neuen Schlag auf dieser Bahn auf dem spezifischen BahnSchild
128             bahnen[currentBahn].schild.addSchlag();
129
130             // Wenn maximale Schläge getätig wurden, irgendwas machen
131             // ...
132
133             club = col.gameObject;
134             club.GetComponent<Collider>().isTrigger = true;
135             GetComponent<Renderer>().material.color = new Color(0, 0.45F, 1F);
136             hasCollided = true;
137             Vector3 diff = (transform.position - col.transform.position);
138             rb.velocity = club.GetComponent<ClubHead>().getVelocity() * 2F;
139         }
140     else if (col.gameObject.CompareTag("Loch")) {
141
142         Debug.Log("Loch getroffen");
143
144         // Setze ggf. den Highscore neu
145         bahnen[currentBahn].schild.updateHighscore();
146
147         if (currentBahn < bahnen.Length - 1) {
148             currentBahn++;
149
150             // Teleport Ball zur nächsten Bahn
151             teleportBall();
152             teleportPlayer();
153         } else {
154             // Setze den Highscore auf dem Schild mit der Gesamtübersicht
155             player.transform.position = new Vector3(0, 0, 29);
156             if ((globalZaehler < globalHighscore || globalHighscore == 0) && globalZaehler > 0) {
157                 globalHighscore = globalZaehler;
158                 setHighScoreText();
159                 PlayerPrefs.SetInt("GesamtHighscore", globalHighscore);
160             }
161
162             // Teleportiere Spieler vor ein Schild mit der Gesamtübersicht und gib ihm
163             // dort die Möglichkeit, sich zurück zur Lobby zu teleportieren.
164         }
165     }
166
167     else if (col.gameObject.CompareTag("Grass")) {
168         // Reset ball to current startpoint
169         teleportBall();
170     }
171 }

```

```

175     private void teleportBall() {
176         rb.velocity = new Vector3(0, 0, 0);
177         rb.angularVelocity = Vector3.zero;
178         transform.position = bahnen[currentBahn].bahnOrigin;
179     }
180
181     private void teleportPlayer() {
182         player.transform.position = bahnen[currentBahn].playerSpawnPoint;
183     }
184
185
186
187     private void setZaehlerText() {
188         zaehlerText.text = "Gesamtschläge: " + globalZaehler;
189     }
190
191     private void setHighScoreText() {
192         if (globalHighscore > 0) {
193             highScoreText.text = "Highscore: " + globalHighscore;
194         } else {
195             highScoreText.text = "Highscore: none";
196         }
197     }
198 }
199

```

Dieses Skript ist auf dem Ball als Component hinzugefügt. Es regelt alles, was mit dem Zählen der Schläge und setzen der Highscores zu tun hat, sowie das Verhalten des Balls im Falle einer Kollision mit dem Schläger oder dem Gras bzw. das Verhalten des Balls, wenn dieser in das Loch fällt. Alle Bahnen werden in dem Array „bahnen“ referenziert. Die Reihenfolge der Objekte bestimmt die Reihenfolge, in welcher die Bahnen bespielt werden.

4. ClubHead.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4
5  public class ClubHead : MonoBehaviour
6  {
7      // Start is called before the first frame update
8      private Vector3 posI;
9      private Vector3 posF;
10     private Vector3 vel;
11     private float velMag;
12     public float maxVel;
13     private Rigidbody _rb;
14
15     private void Awake()
16     {
17         _rb = GetComponent<Rigidbody>();
18     }
19     void Start()
20     {
21         //Store initial and final position at start
22         posI = posF = transform.position;
23     }
24
25     private void FixedUpdate()
26     {
27         //Update initial and final positions
28         posI = posF;
29         posF = transform.position;
30         //_rb.MovePosition(transform.position - (transform.forward * Time.fixedDeltaTime * maxVel));
31         /*
32         if(_rb.velocity != Vector3.zero)
33         {
34             Debug.Log(_rb.velocity + " | " + _rb.position);
35         }
36         */
37     }
38
39     public Vector3 getVelocity()
40     {
41         //Calculate velocity
42         vel = (posF - posI) * 40;
43         velMag = vel.magnitude;
44         //Limit velocity
45         /*if (velMag > maxVel)
46         {
47             vel = vel.normalized * maxVel;
48         }
49         */
50         //Debug.Log("Pos1" + posF + " Pos2" + posI);
51         //Debug.Log(vel * 100);
52         return vel;
53     }
54 }
55 }
```

Dieses Skript ist als Component auf dem Kopf des Golfschlägers hinzugefügt. Es berechnet die Bewegungsrichtung und -geschwindigkeit des Golfschlägers, was bei der Kollision mit dem Golfball von dem Skript „Schlagzaehler.cs“ verwendet wird.

5. SceneTransitionManager.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SceneManagement;
5
6  public class SceneTransitionManager : MonoBehaviour
7  {
8      public FadeScreen fadeScreen;
9
10     //Einfache Methode
11     public void GoToScene(int sceneIndex)
12     {
13         StartCoroutine(GoToSceneRoutine(sceneIndex));
14     }
15
16     IEnumerator GoToSceneRoutine(int sceneIndex)
17     {
18
19         fadeScreen.FadeOut();
20         yield return new WaitForSeconds(fadeScreen.fadeDuration);
21
22         SceneManager.LoadScene(sceneIndex);
23     }
24
25
26     //Bessere Methode bei der nicht alles gleichzeitig geladen wird
27     public void GoToSceneAsync(int sceneIndex)
28     {
29         StartCoroutine(GoToSceneAsyncRoutine(sceneIndex));
30     }
31
32     IEnumerator GoToSceneAsyncRoutine(int sceneIndex)
33     {
34         fadeScreen.FadeOut();
35
36         AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);
37         operation.allowSceneActivation = false;
38         float timer = 0;
39         while(timer <= 0 && !operation.isDone)
40         {
41             timer += Time.deltaTime;
42             yield return null;
43         }
44         operation.allowSceneActivation = true;
45     }
46 }
```

Dieses Skript ist als Component dem Player-Objekt hinzugefügt. Es organisiert den Szenenwechsel, wenn der Spieler von der Lobby zum Rundkurs oder wieder zurück teleportiert wird.

6. FadeScreen.cs

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using UnityEngine.SocialPlatforms;
5
6  public class FadeScreen : MonoBehaviour
7  {
8      public bool fadeOnStart = true;
9      public float fadeDuration = 0;
10     public Color fadeColor;
11     private Renderer rend;
12     public Vector3 dummy;
13     public float count1 = 0;
14
15     void Start()
16     {
17
18         rend = GetComponent<Renderer>();
19         if(fadeOnStart)
20         {
21             FadeIn();
22             this.transform.position = new Vector3(0f, -1f, 0f);
23         }
24     }
25
26     public void FadeIn()
27     {
28
29         Fade(1, 0);
30     }
31
32     public void FadeOut()
33     {
34
35         this.transform.position = dummy;
36         Fade(0, 1);
37     }
38
39
40     public void Fade(float alphaIn, float alphaOut)
41     {
42
43         StartCoroutine(FadeRoutine(alphaIn, alphaOut));
44     }
45
46     public IEnumerator FadeRoutine(float alphaIn, float alphaOut)
47     {
48
49         float timer = 0;
50         while (timer <= fadeDuration)
51         {
52
53             Color newColor = fadeColor;
54             newColor.a = Mathf.Lerp(alphaIn, alphaOut, timer / 1);
55             rend.material.SetColor("_Color", newColor);
56             timer += Time.deltaTime;
57             yield return null;
58         }
59
60         Color newColor2 = fadeColor;
61         newColor2.a = alphaOut;
62         rend.material.SetColor("_Color", newColor2);
63         dummy = this.transform.position;
64     }
65 }
```

Dieses Skript ist als Component auf dem Kamera-Objekt hinzugefügt. Es sollte einen Übergang zu/von Schwarz bei der Teleportation von der Lobby zum Rundkurs oder wieder zurück realisieren. Davon wurde im Rahmen des Testens später allerdings abgesehen und die Übergangsduer deshalb auf 0 gesetzt.

Zusätzliche Abgabe: Blender Datei:

Vorwort:

Als zusätzliche Abgabe haben wir noch eine Blender Datei hinterlegt, welche das Erstellen von Golfbahnen erleichtert. Die Erstellung war recht aufwändig, weshalb es schade wäre die Datei nicht mitabzugeben.

Die Blender Datei enthält eine Collection namens „Assets“, welche verschiedene Segmente einer Golfbahn enthält. Eine Bahn kann dann durch das Zusammenfügen der Segmente erstellt werden und man kann natürlich auch noch eigene zusätzliche Elemente hinzu modellieren. Im Folgenden werden die einzelnen Segmente mit ihren veränderlichen Eigenschaften beschrieben.

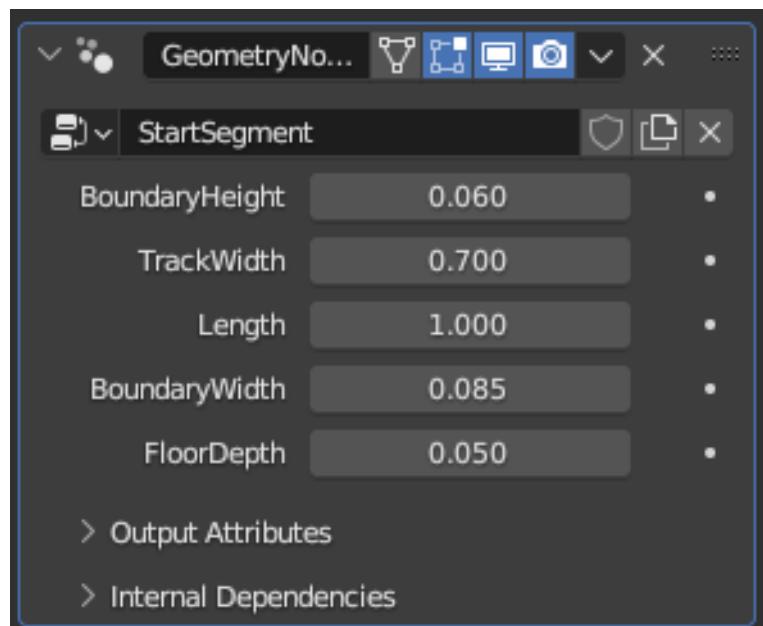
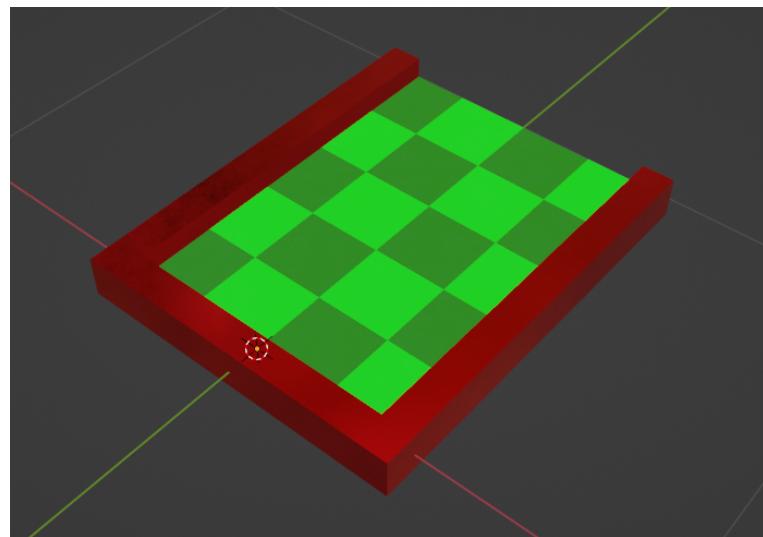
Wichtige Information:

Bei jedem Segment kann man die „BoundaryHeight“ (Bandenhöhe), „BoundaryWidth“ (Bandenbreite), „TrackWidth“ (Streckenbreite), „Length“ (Länge) und die „FloorDepth“ (Bodentiefe) beliebig verändern bzw. die Größe / Skalierung einstellen.

Im Folgenden werden deswegen nur die anderen veränderbaren Werte beschrieben.

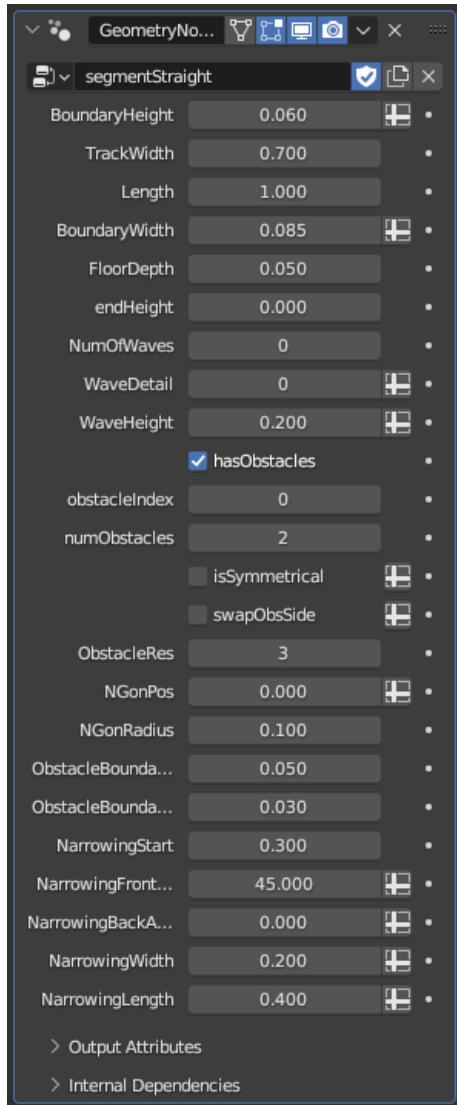
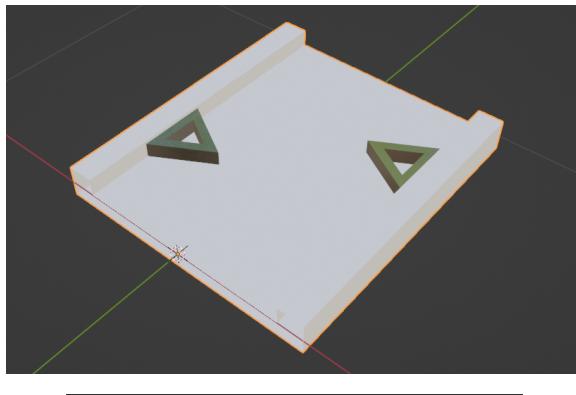
Start Segment:

Das „StartSegment“ bildet den Anfang einer Bahn oder ist als Eckstück vorhanden und es sind lediglich die zuvor genannten Werte veränderlich.



Straight Segment:

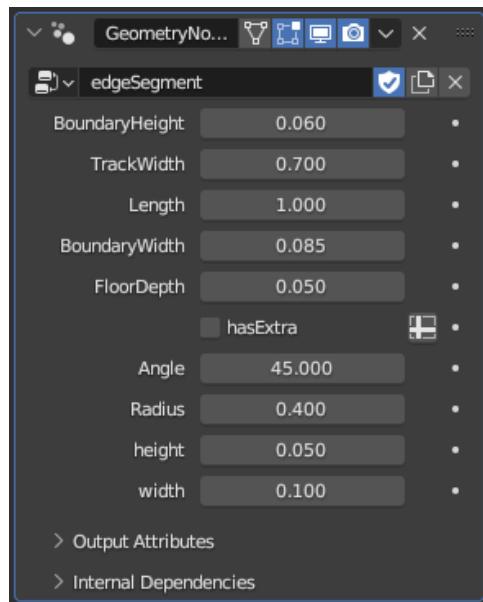
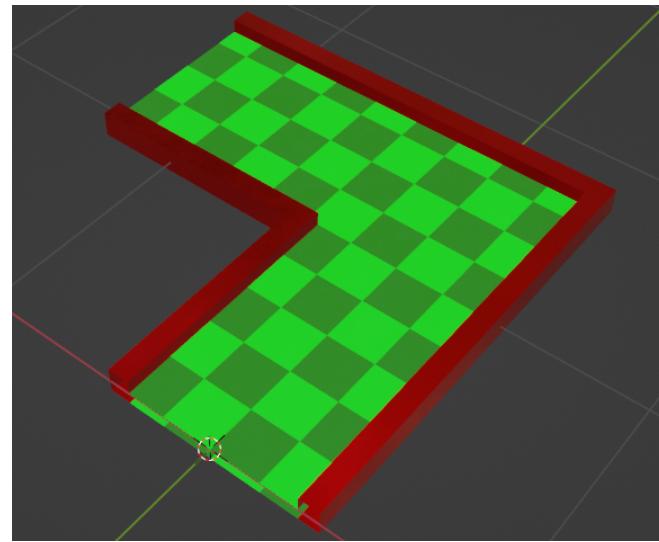
Das „StraightSegment“ ist ein gerades Verbindungsstück, welches viele Option mit sich bringt. Das Segment kann Hindernisse enthalten, wenn die Checkbox „HasObstacles“ ausgewählt ist, die „Endheight“ null ist und „NumOfWaves“ null ist.



- „endHeight“ gibt den Höhenunterschied zwischen Start und Ende der Bahn an
- „NumOfWaves“ gibt die Anzahl der Wellen des Segments an
- „WaveDetail“ gibt an, wie detailliert die Rundung der Wellen ist
- „WaveHeight“ gibt die Höhe der Wellen an
- „HasObstacles“: Wenn die Checkbox „HasObstacles“ ausgewählt ist, die „Endheight“ null ist und „NumOfWaves“ null ist, kann das Segment Hindernisse enthalten
- „obstacleIndex“: Es gibt verschiedene Obstacles, welche über diesen Index ausgewählt werden können
- „numObstacles“ gibt die Anzahl der Obstacles an
- „isSymmetrical“ ordnet die Obstacles symmetrisch an, wenn ausgewählt
- „swapObsSide“ Spiegelung der Obstacle Anordnung
- „ObstacleRes“ Anzahl der Ecken des Obstacles (gleichseitiges Vieleck)
- „NGonPos“ ist das Offset entlang der Bahn (Verschiebung der Obstacles)
- „NGonRadius“ gibt den Radius des Obstacles an
- „ObstacleBoundaryHeight“ gibt die Höhe des Obstacles an
- „ObstacleBoundaryWidth“ gibt die Breite des Obstacles an
- „NarrowingStart“ gibt das Offset für Narrowing an
- „NarrowingFrontAngle“ ist der Winkel des vorderen Teils des Obstacles
- „NarrowingBackAngle“ ist der Winkel des hinteren Teils des Obstacles
- „NarrowingWidth“ gibt an wie weit das Obstacle in die Bahn ragt
- „NarrowingLength“ gibt die Länge des Obstacles an

Edge Segment:

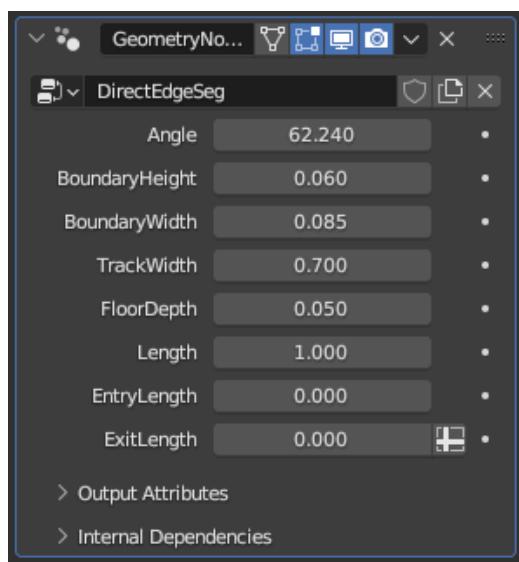
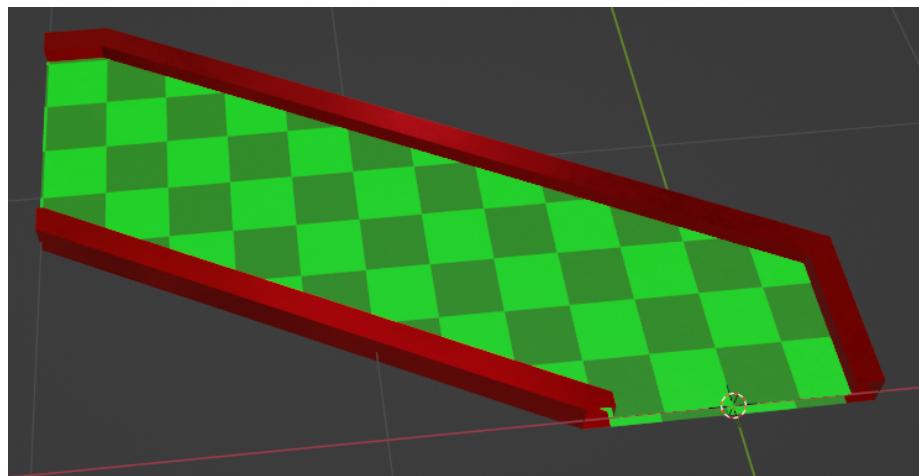
Das „EdgeSegment“ ist ein 90° Winkel bzw. ein Eckstück der Bahn. Dieses kann noch ein extra Element im Eck haben.



- „Angle“ ist der Winkel des extra Elements
- „Radius“ gibt die Tiefe der Einrückung an
- „height“ gibt die Höhe des extra Elements an
- „width“ gibt die Breite des extra Elements an

Direct Edge Segment:

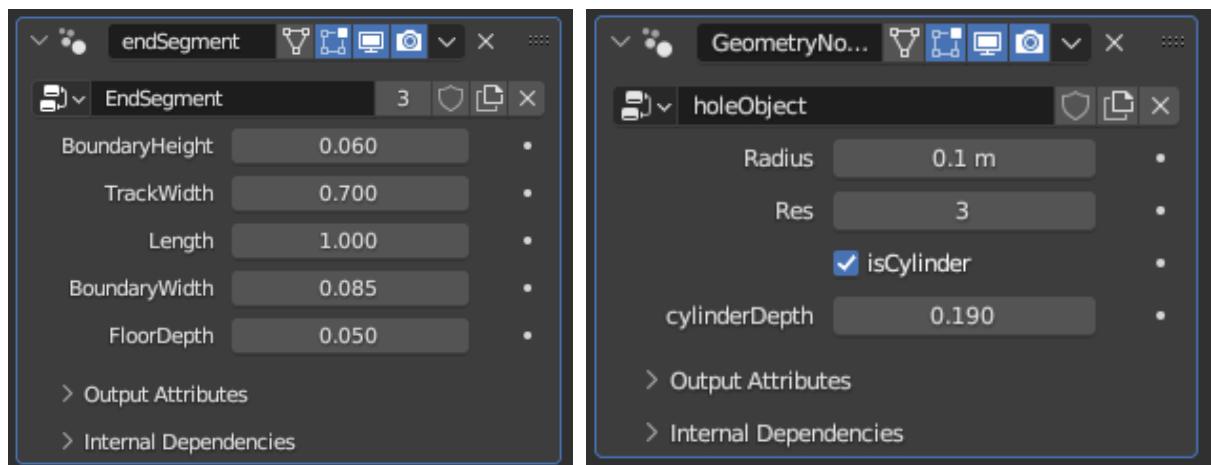
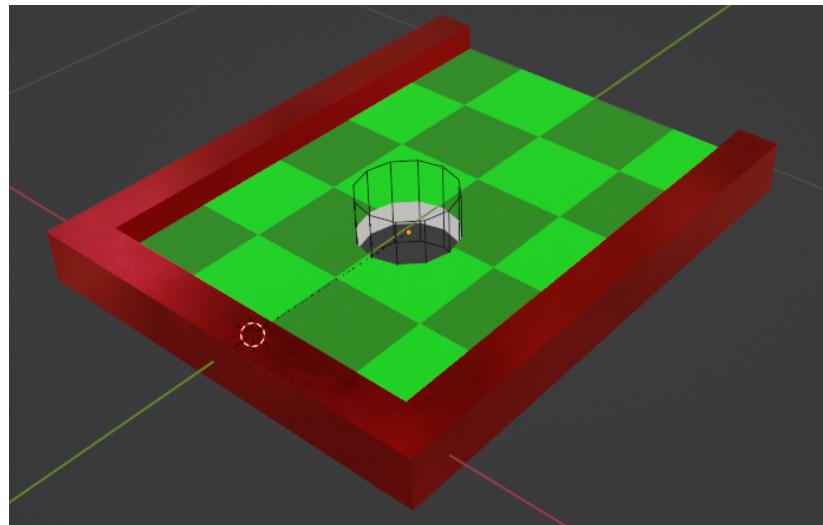
Das „**DirectEdgeSegment**“ ist ein Eckstück, wobei man den Winkel verstellen kann. Des Weiteren können eine „**EntryLength**“ und „**ExitLength**“ eingestellt werden.



- „Angle“ gibt den Winkel an
- „Length“ gibt die Distanz der Mittelpunkte von Anfang und Ende des Segments an
- „EntryLength“ gibt an, wie lange die gerade Stecke vor dem Winkel sein soll
- „ExitLength“ gibt an, wie lange die gerade Strecke nach dem Winkel sein soll

End Segment:

Das „EndSegment“ bildet das typische Ende der Bahn und enthält somit auch das Loch. Das „EndSegment“ hat ein Kindobjekt das „HoleObject“. Aus dem „EndSegment“ wird mit Hilfe des „HoleObject“ das Loch ausgeschnitten. Hierbei ist Folgendes variabel:

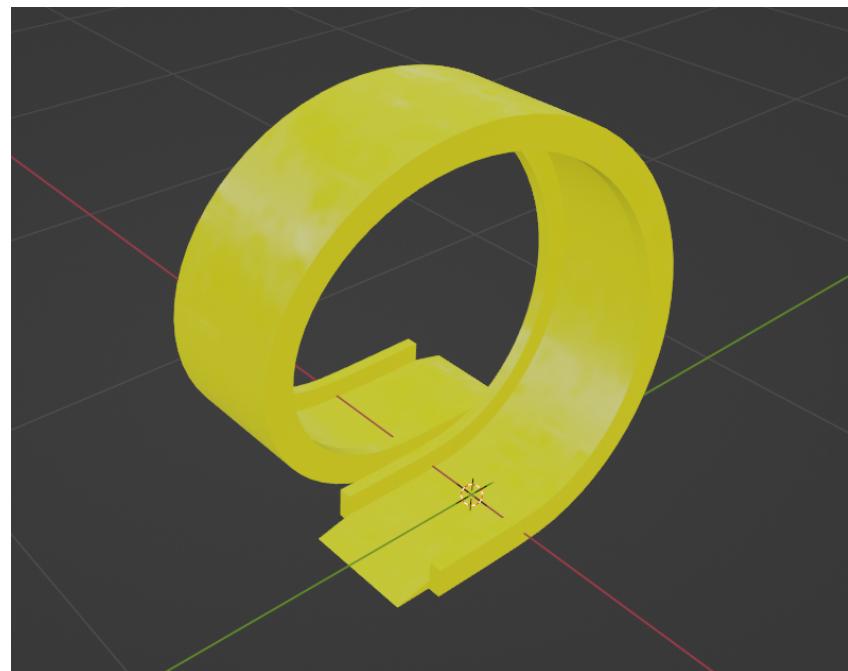


- „Radius“ gibt den Radius an
- „Res“ gibt den Grad der Rundung an
- „isCylinder“ gibt, wenn ausgewählt die Form eines Zylinders, wenn nicht, die Form einer Kugel
- „cylinderDepth“ gibt die Höhe des Zylinders an, falls zuvor ausgewählt

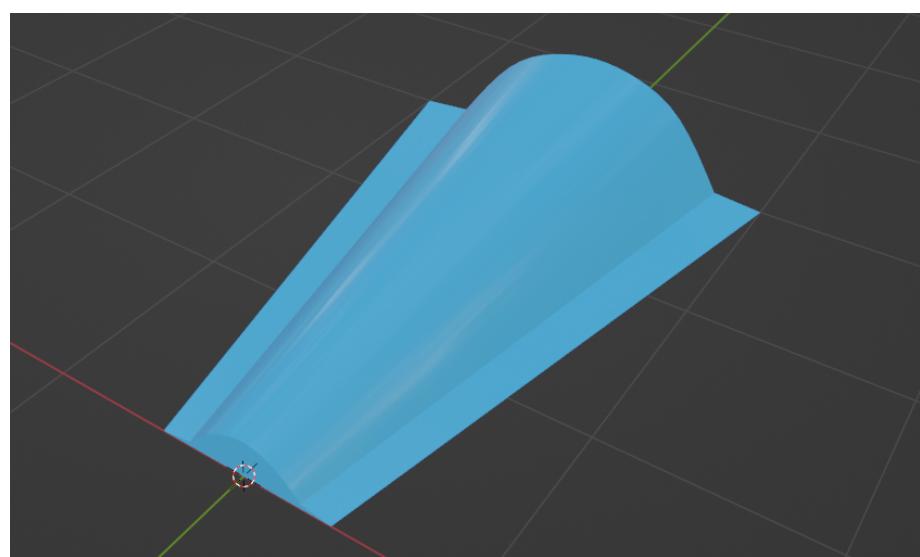
Obstacles:

Neben Der Collection „Assets“ existiert auch eine Collection names „Obstacles“, welche typische Hindernisse einer Minigolfbahn enthält.

Das erste Element ist ein „Looping“:



Das zweite Element sind „RoundObs“:



Erstellen einer Bahn:

Diese Segmente sollte man nun passend zusammenlegen, sodass die Kanten aufeinanderliegen.

Der GeometryNodes Modifier muss nun für jedes Segment applied werden. Danach muss man die Segmente joinen, sodass man genau ein Objekt erhält.

Als nächstes wechselt man in den Edit Mode, wählt alles aus und führt die Aktion „Merge by Distance“ aus.

Sollte man selbst gemodelte Objekte oder das „StraightSegment“ verwendet haben, so muss man diesen Elementen die Materials selbst zuweisen. Ansonsten sind die Materials bereits vorgefertigt.

Bei runden Elementen sollte man noch die Funktion „Shade Auto Smooth“ verwenden.

Am Ende sollte man das „HoleObject“ beim „EndSegment“ löschen.