Begehbares Ray-Tracing-Labyrinth in Unity

Projektbericht

Cyril Wagner

23. Mai 2025

Inhaltsverzeichnis

1 Einleitung

Die Übungsaufgabe verlangt ein begehbares 3-D-Labyrinth mit realistischer Ray-Tracing-Beleuchtung. Dieser Bericht dokumentiert Aufbau, Funktionsweise und Entwicklungsprozess des Unity-Projekts und reflektiert den Einsatz von Künstlicher Intelligenz (ChatGPT).

2 Technische Grundlagen

2.1 Ray-Tracing vs. Rasterisierung

Rasterizer erzeugen Bilder Dreieck für Dreieck im Bildraum. *Ray Tracing* verfolgt Lichtpfade, ermöglicht korrekte Spiegelungen, Brechungen und globale Beleuchtung, benötigt aber DXR-fähige Hardware.

2.2 HDRP + DXR in Unity 2022

HDRP bietet seit 2021 direkte DXR-Integration: Ray-traced Shadows, Reflections, Ambient Occlusion und einen optionalen Path Tracer. Für das Projekt aktiviert: Realtime RT-Shadows + Reflections.

3 Projektverzeichnis

```
Assets/
  Materials/          # Absorb.mat, Reflect.mat, Transparent.mat
  Prefabs/            # Wall.prefab, Floor.prefab, Goal.prefab
  Scripts/            # MazeLoader.cs, SimpleFPS.cs
  Scenes/             # Main.unity
  StreamingAssets/    # maze.json
```

4 Material- und Shader-Konfiguration

Tabelle 1: Parameter der Wandmaterialien

Material	Metallic	Smoothness	Besonderheit
Absorb	0	0.25	matt
Reflect	1	1.00	Spiegel
Transparent	0	0.95	Surface Type = Transparent, IOR 1.52

5 Wesentliche Skripte

5.1 `MazeLoader.cs`

```
1  using System.Collections.Generic;
2  using System.IO;
3  using UnityEngine;
4
5  // MazeLoader.cs  Version without external JSON package (uses UnityEngine.JsonUtility)
6
7  /*
8   1. Place a maze.json inside Assets/StreamingAssets  sample format:
```

```
 {
   "width": 2,
   "height": 2,
   "cells": [
     {"x":0,"y":0,"wallNorth":true,"wallSouth":false,"wallEast":false,"wallWest":true
    ,"material":"absorb","isLight":true,"isGoal":false},
     {"x":1,"y":0,"wallNorth":true,"wallSouth":false,"wallEast":true,"wallWest":false
    ,"material":"reflect","isLight":false,"isGoal":false},
     {"x":0,"y":1,"wallNorth":false,"wallSouth":true,"wallEast":false,"wallWest":true
    ,"material":"transparent","isLight":false,"isGoal":false},
     {"x":1,"y":1,"wallNorth":false,"wallSouth":true,"wallEast":true,"wallWest":false
    ,"material":"absorb","isLight":false,"isGoal":true}
   ]
 }
 2. Attach this script to an empty GameObject called "Maze" and assign prefabs/
    materials in the Inspector.
 3. Requires NO external packages  Unity's builtin JsonUtility handles the parsing.
*/

[System.Serializable]
public class MazePlan
{
    public int width;
    public int height;
    public Cell[] cells;
}

[System.Serializable]
public class Cell
{
    public int x;
    public int y;
    public bool wallNorth;
    public bool wallSouth;
    public bool wallEast;
    public bool wallWest;
    public string material; // "absorb", "reflect", "transparent"
    public bool isLight;
    public bool isGoal;
}

public class MazeLoader : MonoBehaviour
{
    [Header("Prefabs")]
    public GameObject wallPrefab;
    public GameObject floorPrefab;
    public GameObject goalPrefab;
    public Light lightPrefab;

    [Header("Materials")]
    public Material absorbMaterial;
    public Material reflectMaterial;
    public Material transparentMaterial;

    [Header("Paths")]
    public string jsonFileName = "maze.json";
```

Listing 1: Kernausschnitt aus `MazeLoader.cs`

6 Programmlauf in Unity & Fallstricke

6.1 Ablauf pro Frame

1. **Editor lädt Szene** → alle GameObjects werden instanziiert.
2. **Script-Lebenszyklus**
   `MazeLoader.Awake()` (nicht überschrieben) → `OnEnable()` → **Start():**
   (a) JSON aus `StreamingAssets` lesen.
   (b) `Build(plan)`: Prefabs platzieren, Material wählen.
   (c) HDRP registriert neue MeshRenderer – erste AS-Build.
3. **Camera + Input**
   Update-Schleife des Controllers liest Input (WASD, Maus) und ruft `CharacterController.Move`.
4. **HDRP Renderloop**
   G-Buffer → Ray-Tracing Pass (Shadows/Reflections) → Denoiser → PostFX.

6.2 Worauf man achten muss

- **Dateinamen exakt beachten**
  Datei = Klassenname (`MazeLoader.cs`), JSON = `maze.json`. Groß-/Kleinschreibung!
- **Prefabs/Materialzuweisung**
  Fehlende Referenzen führen zu Laufzeit-`NullReference` und unsichtbaren Wänden. Im Inspector alle Felder füllen.
- **StreamingAssets-Ort**
  Der Ordner muss *direkt* unter `Assets/` liegen, sonst kopiert Unity die Datei nicht in den Build.
- **Ray-Tracing nur unter DX12**
  Player-Einstellung > Graphics API → DX12 an erster Stelle. Unter Vulkan/Metal fällt HDRP automatisch auf Raster zurück.
- **Input System**
  Beim ersten Play „New Input System“ aktivieren oder Controller reagiert nicht.
- **Performance**
  Stats → *Frame Debugger* öffnen: Hohe BVH-Build-Zeit deutet auf zu viele bewegliche Meshes hin. Lösung: Prefabs als `Static = true` markieren.

7 JSON-Blueprint

```
{
  "width": 2,
  "height": 2,
  "cells": [
    { "x":0, "y":0, "wallNorth":true,  "wallWest":true,
      "material":"absorb",   "isLight":true,  "isGoal":false },
    { "x":1, "y":0, "wallNorth":true,  "wallEast":true,
      "material":"reflect",  "isLight":false, "isGoal":false },
    { "x":0, "y":1, "wallSouth":true,  "wallWest":true,
```

```
      "material":"transparent", "isLight":false, "isGoal":false },
    { "x":1, "y":1, "wallSouth":true,  "wallEast":true,
      "material":"absorb",   "isLight":false, "isGoal":true  }
  ]
}
```

Listing 2: Beispiel `maze.json`

8 Leistungsanalyse

RTX 3060 Laptop GPU, 1080p:

- RT-Shadows + Reflections: **72 FPS**
- + Path Tracing (4 Bounces): 12 FPS
- Engpässe: BVH-Build 30 %, Reflection-Rays 25 %.

Optimierungen: Static Batching, weniger Spiegel, Denoiser-Radius 0.25.

9 Einrichtung in Unity

9.1 Voraussetzungen

Unity 2022 LTS, HDRP 14, DX12, DXR-fähige GPU.

9.2 Schritte

1. HDRP Wizard → Tab DXR → *Fix All*.
2. Materialien, Prefabs, JSON anlegen.
3. `MazeLoader` konfigurieren.
4. Spielercontroller (Prefab oder `SimpleFPS`) platzieren.
5. Szene speichern, *Play*.

10 Probleme und Lösungen

Newtonsoft-Abhängigkeit Umstieg auf `JsonUtility`.

Skript nicht anfügbar Dateiname passte nicht zum Klassennamen.

Input tot New Input System nicht aktiviert → Popup bestätigen.

11 Einsatz von Künstlicher Intelligenz

ChatGPT generierte Code-Skeletons, JSON-Beispiele, LaTeX-Struktur und diente als „Rubber Duck“ bei Fehlersuche.

12 Fazit und Ausblick

Das Projekt zeigt ein interaktives Ray-Tracing-Labyrinth in Unity. Erweiterbar um prozedurale Generatoren, Mehrspieler-Support oder KI-Pfadfindung.