

Übung 1

1 Hardware und Game Engine

Die Lösung zur ersten Übungsaufgabe wurde auf einem Notebook mit Intel Core i9 13900HX CPU, 32 GB RAM und NVIDIA GeForce RTX 4070 GPU erstellt. Als Game Engine wurde Unity verwendet. Die Auswahl fiel auf diese Engine, da der Einstieg vergleichsweise einfach sein soll und C# als Programmierapache Verwendung findet[[Glit24](#)]. Außerdem wird in der High Definition Render Pipeline (HDRP) Ray Tracing unterstützt [[Unit25](#)]. Unity verfügt zudem über ein grafisches User Interface (UI) und aufgrund der Verbreitung findet man viele Tutorials und Dokumentationen.

2 Implementierung

Als Ausgangspunkt wurde das von Unity bereitgestellte High Definition 3D sample v16.0.11 Beispielprojekt verwendet. Dieses beinhaltet auch ein Third Person Controller mit Charakter und Animation, welcher als Prefab für den Spieler übernommen wurde. Zudem das Lighting sowie das globale Volume übernommen. Die Szene wurde in `SampleScene` implementiert und ist unter `Assets/Scenes/` zu finden. Das Ray Tracing wurde anhand eines Youtube Tutorials aktiviert¹. Bei der Lösung der Aufgabe wurde ChatGPT² zur Generierung von Codefragmenten und Erläuterung von Unity Funktionalitäten verwendet.

2.1 Assets

Um die Spielwelt zu generieren wurden folgende zusätzlichen Assets erstellt:

1. Materialen für die Umgebung im Ordner `Assets/Materials/General/`

- (a) GlassMaterial1: transparentes Material, das ein blaues Glaselement darstellen soll.

¹https://www.youtube.com/watch?v=ad9f_nKU0ZA

²<https://chatgpt.com/>

- (b) MirrorMaterial2: spiegelndes Material, mit den Einstellungen Metallic=1 und Smoothness=1
 - (c) FloorMaterial3: absorbierendes Material für die Wände und den Boden
 - (d) GlowMaterial: emittierendes Material für die Lampen
 - (e) GoalMaterial: glänzend-goldenes Material für das Zielobjekt, das Skript `GoalTrigger.cs` startet die Szene neu, sobald das Objekt berührt wird
2. Gegenstände im Ordner **Assets/Prefab**

- (a) GoalBall: stellt das Zielobjekt dar
- (b) Lamp: Standlicht mit Punktbeleuchtung

2.2 Labyrinth

Das Labyrinth wird anhand einer Bitmap vollständig generiert. Das Labyrinth wird als `maze.bmp` im Ordner **Assets/Resources** gespeichert. Das Skript `LabyrinthParser.cs` im Ordner **Assets/Scripts**, welches an `GameObject Maze` der Szene hängt, verarbeitet die Bitmap Pixel für Pixel und generiert das Labyrinth nach folgendem Schema:

- RGB(255,255,255) Weiß: Boden
- RGB(0,0,0) Schwarz: absorbierende Wand
- RGB(0,255,0) Grün: transparente Wand
- RGB(255,0,0) Rot: spiegelnde Wand
- RGB(0,0,255) Blau: Spawnpunkt Spieler
- RGB(255,255,0) Gelb: Lampe
- RGB(255,0,255) Magenta: Ziel

Die Vorlage des verwendeten Labyrinths wurde mit Hilfe von `mazegenerator`³ erstellt.

³<https://mazegenerator.net/>

2.3 Beleuchtung

Zur Beleuchtung der Umgebung wurde das globale Lighting des Beispielprojekts übernommen. Hierin enthalten ist ein gerichtetes Sonnenlicht. Als zusätzliche Beleuchtung dienen im Labyrinth die Lampen in Form von Punktlichtern. Aufgrund der teils umgebenden reflektierenden Materialien wurde die Anzahl an Reflexionen auf vier gesetzt. Bei dieser Anzahl war die Performanz noch ausreichend bei ca. 15 - 20 Frames im Editor. Abbildung 1 stellt dar, dass ab dieser Anzahl von vier Reflexionen Flächen nur noch Schwarz dargestellt werden.

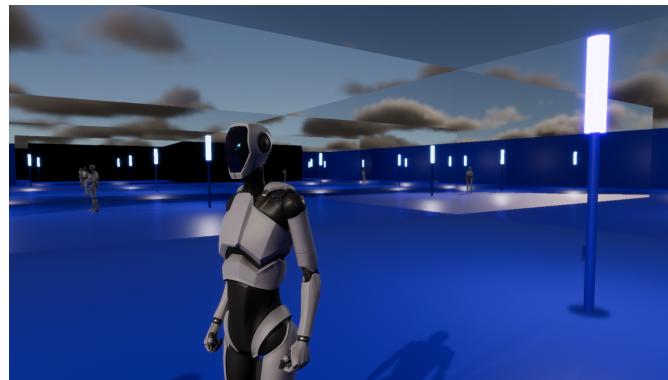


Abbildung 1: Spiegelungen im Labyrinth mit schwarzen Flächen im Hintergrund.

3 Reflexion

Im Rahmen der Übung konnten grundlegende Konzepte des Spieleentwicklung in Unity erlernt werden. Zuvor waren keine Kenntnisse bei der Bedienung der Engine. Die optimale Einstellung der Beleuchtung stellte bei der Entwicklung einen iterativen Prozess dar, der nicht trivial erschien. Durch zahlreiches Ausprobieren konnten optisch akzeptable Ergebnisse erzielt werden. Ebenso verhielt es sich im Feintuning des Ray Tracings. Beispielsweise aufgrund der Anzahl an Reflexionen war die Framerate im kompilierten Spiel im einstelligen Bereich. Um die Eigenschaften der unterschiedlichen Materialien und den Umgang zu erlernen wurden diese selbst erstellt. Da keine weiteren Strukturen oder Shader verwendet wurden, wirkt sich dies jedoch auf die Optik aus.

Literatur

- [Glit24] Glitch Guru. Game Engines: A Comparative Analysis. <https://medium.com/@Glitch-Guru/game-engines-a-comparative-analysis-ef9af01f125e>, 2024. Stand: 20.01.2024, Aufruf am: 24.05.2025.
- [Unit25] Unity. Getting started with ray tracing. <https://docs.unity3d.com/Packages/com.unity.render-pipelines.high-definition@9.0/manual/Ray-Tracing-Getting-Started.html>, 2025. Stand: 24.05.2025, Aufruf am: 24.05.2025.