Spieleprogrammierung
Übung 1

Tillmann Faust

25.05.2025

Vorüberlegungen

Für die Umsetzung dieses Projekts wurde sich für Unity entschieden. Das Projekt wurde als leeres HDRP Projekt gestartet und anschließend darauf aufgebaut. Ziel des ”Spiels” ist es, aus dem Labyrinth heraus zu finden.

Umsetzung

Das Labyrinth

Speicherformat

Zum speichern des Labyrinths wurde sich für eine JSON-Datei der Form

```
{
  "walls":[
    {
      "x1":x1,
      "y1":y1,
      "x2":x2,
      "y2":y2,
      "type":material_type
    },
    {
      ...
    },
    ...
  ]
}
```

entschieden. Jeder Eintrag der Liste ”walls” codiert eine Wand und besteht aus den Start- und Endkoordinaten auf einer zweidimensionalen Ebene und einer Kennung für das zu verwendende Material. Die akzeptierten Optionen sind dabei ”transparent”, ”reflective” und ”absorbent”. Die JSON-Dateien befinden sich im Unity-Projekt unter ’Assets/Ressources/’. Die für die finale Abgabe verwendete Datei ist ’labyrinth2.json’.

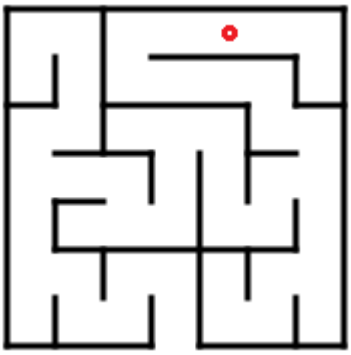

Abbildung 1: Das verwendete Labyrinth (Der Startpunkt ist rot markiert)

MazeLoader

Um aus einer gegebenen Codierung des Labyrinths ein tatsächliches Labyrinth zu generieren wurde das Skript MazeLoader.cs geschrieben. Im groben nimmt das Skript die angegebene JSON-Datei, extrahiert die einzelnen Einträge für jede Wand und instantiiert diese an der richtigen Position und mit richtigen Längen-, Breiten- und Höhenwerten. Zusätzlich zur zu verwendenden JSON-Datei können die gewünschte Höhe und Breite der einzelnen Wände, ein Wert für die Skalierung der ursprünglichen Grid-Koordinaten, sowie die drei Materialien, welche für die Wände verwendet werden sollen von außen übergeben werden.
Während der Entwicklung kam es zu dem Problem, dass überlappende Objekte verschiedener Materialien für graphische Probleme sorgen können. Daher wurde die Länge der generierten Wände um die Wandbreite an jeder Seite verkürzt.

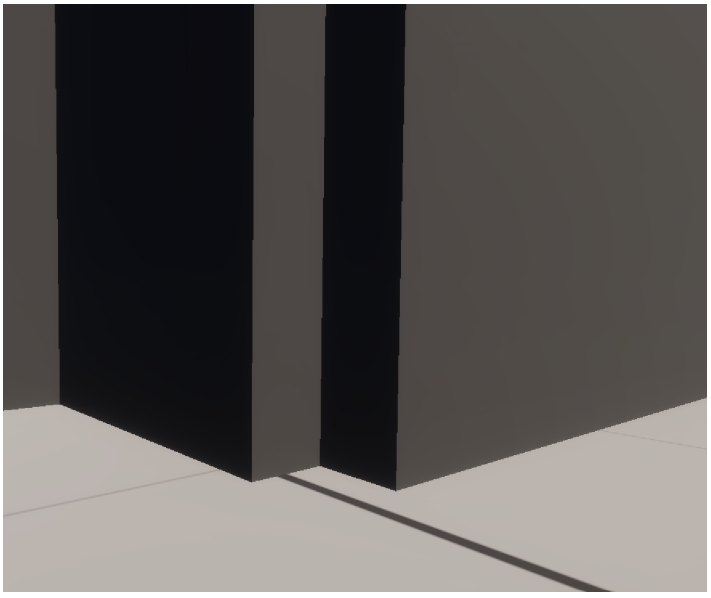

Abbildung 2: Lücken zwischen Wänden

Dies hatte Lücken zwischen einzelnen Wänden zur Folge, weswegen zusätzlich an jedem Start und Endpunkt jeder Wand eine Säule mit einem Radius von $2 * thickness$ platziert wird. Die Säulen verwenden das absorbierende Material und maskieren die Lücken zwischen den Wänden.

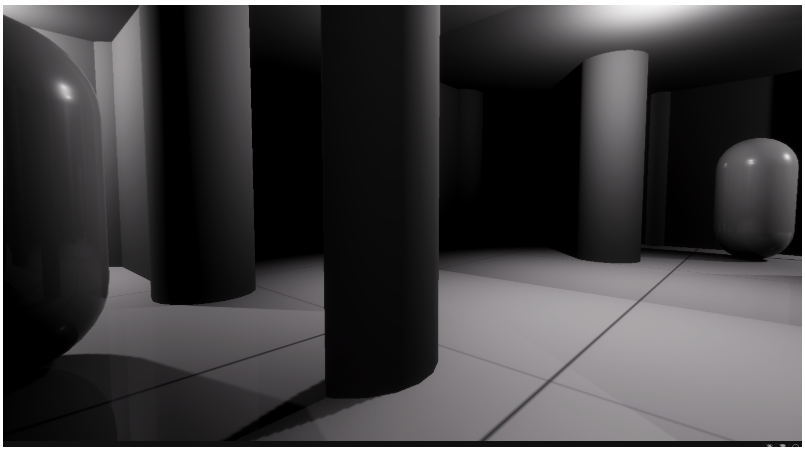

Abbildung 3: Das Labyrinth von innen, mit Spiegeln und Säulen

Zuletzt wird ein Dach auf das Labyrinth gesetzt um alles abzuschließen. Die Größe des Daches ist zum aktuellen Zeitpunkt hart-codiert.

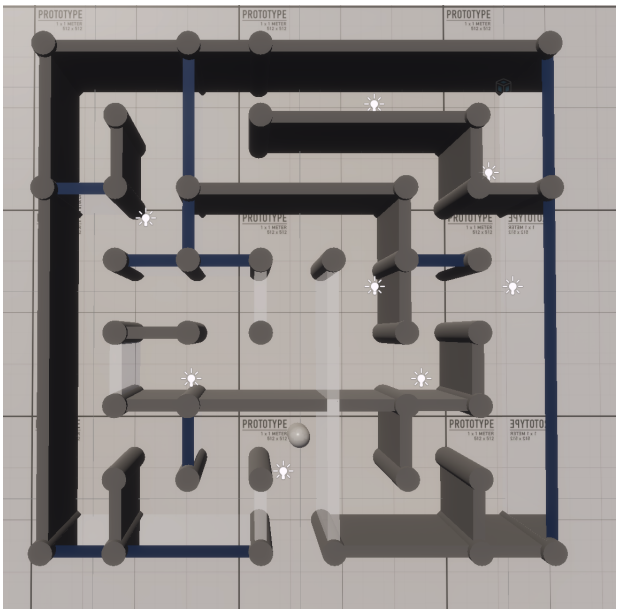

Abbildung 4: Das Labyrinth von oben

Die Spielwelt

Die Spielwelt besteht aus einer Plane als Boden, dem Spielercharakter, dem durch den MazeLoader generierten Labyrinth, sowie einigen Lichtquellen. Zwar wird das Labyrinth aus der JSON-Datei generiert, die Lichter selbst wurden jedoch manuell platziert, weswegen die Verwendung einer anderen Labyrinth-Schablone zu Problemen führen kann. Es ist zu überlegen, ob man die Platzierung der Lichter mit in die JSON-Datei aufnimmt.

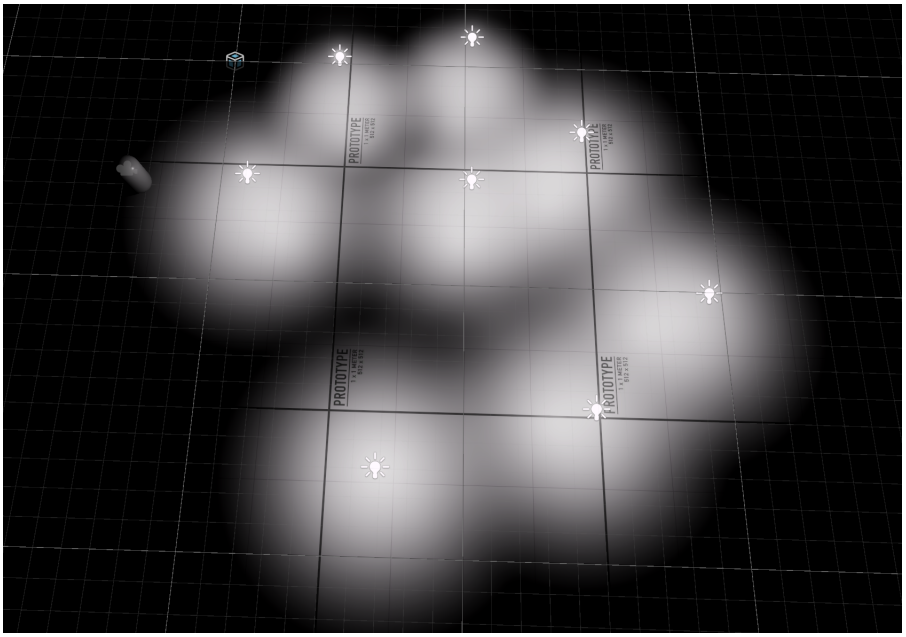

Abbildung 5: Die Spielwelt ohne Labyrinth

Der Boden in den Screentshots verwendet eine kostenlose Textur aus dem Asset-Store (https://assetstore.unity.com/packages/2d/textures-materials/gridbox-prototype-materials-129127). Diese Textur unterliegt der Standard Unity Asset Store EULA, darf meines Wissens nach daher nicht mit den Projektdateien abgegeben werden und fehlt entsprechend in der Abgabe.

Der Spieler

Der Spielercharakter besteht aus zwei Komponenten, der Kamera und dem Körper sowie drei Skripten, welche Bewegung von Kamera und Körper sowie die Positionierung der Kamera zuständig sind. Es wurde eine Anleitung unter "https://www.youtube.com/watch?v=f473C43s8nE" befolgt. Da die Anleitung bereits drei Jahre alt ist wird allerdings noch das alte Input System verwendet, was von Unity angemerkt, bzw. angemängelt wird.

Raytracing

Der wichtigste Punkt der Aufgabe war das Raytracing, dessen Aktivierung einige Einstellungen und Schritte erforderten. Zu beginn musste das Projekt direkt

als HD Render Pipeline Projekt gestartet werden, da eine Umwandlung von URP zu HDRP nicht möglich war. Anschließend musste in den Einstellungen des richtigen HDRP-Assets - Unity generiert standardmäßig mehrere, u.a. für verschiedene Performance Optionen - Raytracing für licht und Schatten, sowie Realtime Raytracing aktiviert werden. In den Projekteinstellungen für Grafik und Spieler mussten ebenfalls zusätzliche Einstellungen vorgenommen werden, damit richtiges Raytracing verwendet wird und damit insbesondere auch Spiegelnde Oberflächen nicht nur die Spiegelung durch RT erzeugen, sondern eingehendes licht auch tatsächlich reflektieren. Das letzte Problem, das gelöst werden konnte war, dass Licht durch die Übergänge zwischen Boden/Dach und Wänden "bluten" konnte. Dies wurde durch die Verwendung von Shadowmaps unterbinden.

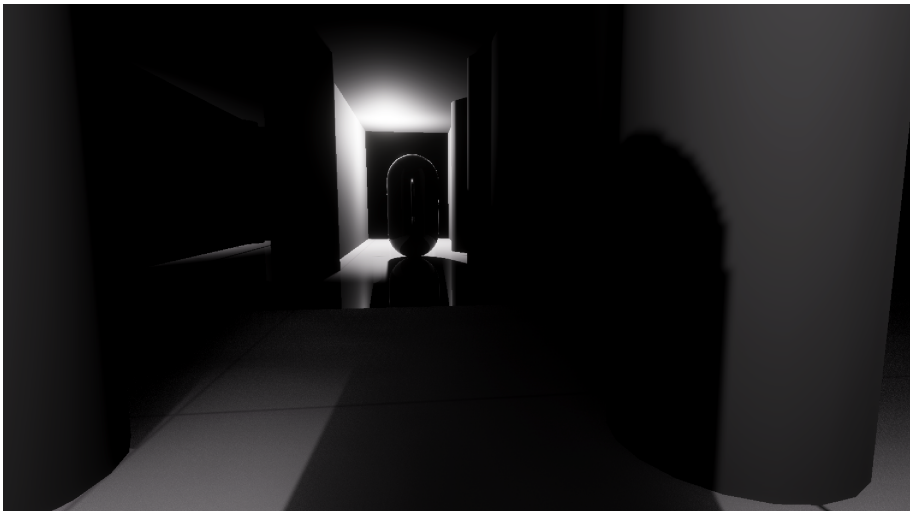

Abbildung 6: Spiegelung, Licht und Schatten