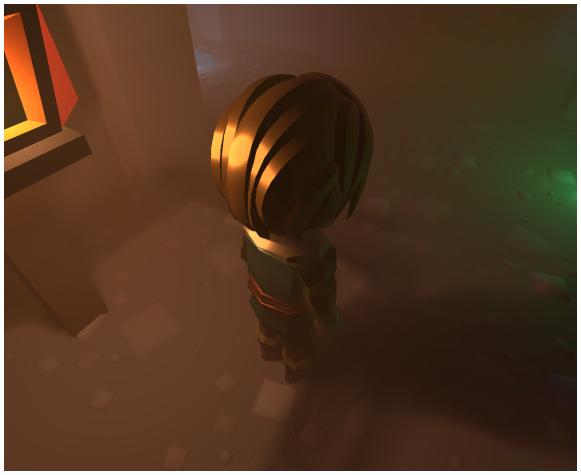
Spieleengine

Godot 4.4

Für die Umsetzung des Programmierprojekts habe ich Godot 4.4 verwendet. Entscheidend für diese Entscheidung war, dass ich mobil an einem Laptop mit einem Arch Linux Betriebssystem entwickle. Godot liefert diesbezüglich eine leichtgewichtige Engine, geringe Hardwareanforderungen und einen performanten Editor für Linux-Systeme.[1] Zudem bin ich in der Spieleentwicklung noch Anfänger und wollte eine Engine mit geringer Lernhürde nutzen. Ein weiterer Vorteil ist, dass Godot eine vollständige Open-Source Software ist und unter der MIT-Lizenz publiziert wird.[2]



Shadow Mapping

Godot 4.4 setzt im Echtzeitbetrieb auf Shadow Mapping statt auf Ray Tracing.[3] Dabei wird die Szene aus Sicht einer OmniLight3D Lichtquelle in eine Tiefenkarte (Shadow Map) gerendert und im Videospeicher gehalten. In jedem Frame erfolgt dann nur noch ein Tiefenvergleich, um zu entscheiden, ob ein Pixel im Schatten liegt oder nicht.[4] Dieses Verfahren ist deutlich weniger rechenintensiv als vollständiges Ray Tracing und erlaubt flüssige Bildraten auch auf weniger leistungsfähiger Hardware.

Screen-Space Reflections

Wird der Forward+ Renderer verwendet, können GPU-beschleunigte Screen-Space Reflections (SSR) eingesetzt werden.[5] Die Reflexionen basieren dann auf Cubemap-Texturen, die von Reflection Probes erzeugt werden: Jeder Probe rendert eine konfigurierbare Umgebung in eine Cubemap, die SSR direkt für dynamische Spiegelungen auf glatten Oberflächen nutzt.[6] Godot bietet dabei zwei Modelle: statische und dynamische Reflection

[1] RocketBrush Blog. *Godot vs Unity Which Game Engine Should You Choose?* URL: `https://rocketbrush.com/blog/godot-vs-unity`.

[2] *Godot Engine License.* URL: `https://godotengine.org/license/`.

[3] *3D lights and shadows Godot Engine Documentation (4.4).* URL: `https://docs.godotengine.org/en/4.4/tutorials/3d/lights_and_shadows.html`.

[4] *Shadow mapping.* URL: `https://en.wikipedia.org/wiki/Shadow_mapping`.

[5] *Environment and post-processing Godot Engine Documentation (4.4).* URL: `https://docs.godotengine.org/en/4.4/tutorials/3d/environment_and_post_processing.html`.

[6] *Screen Space Reflections.* URL: `https://de.wikipedia.org/wiki/Screen_Space_Reflections`.

Probes. Im statischen Modus die Cubemap einmal beim Szenenstart initialisiert und im Videospeicher verwaltet, während für den dynamischen Modus Neuberechnungen in jedem Frame durchführt werden.[7]

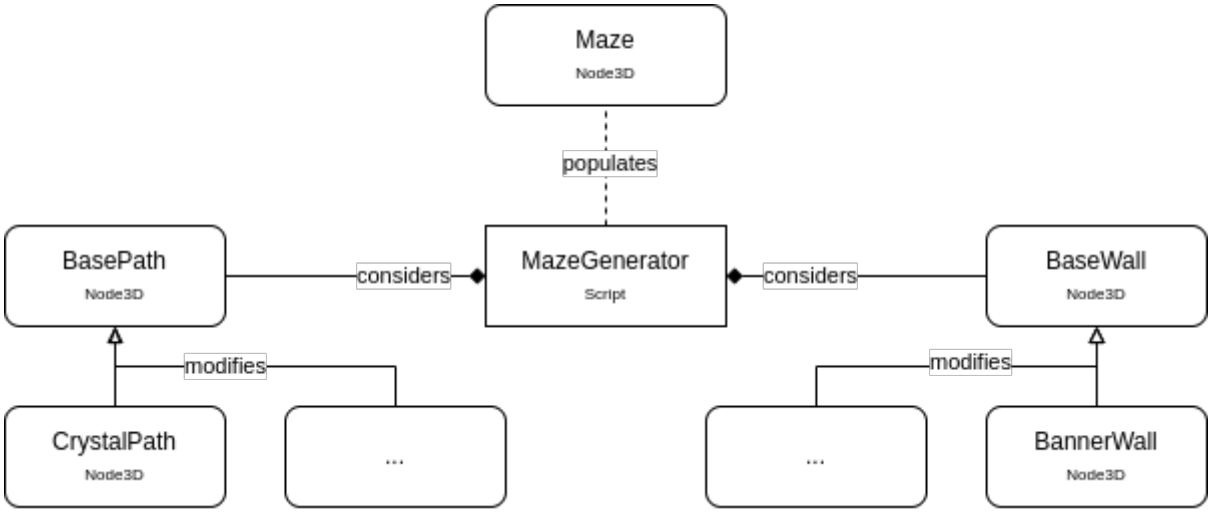
GDScript

Godot setzt auf GDScript, eine dynamisch typisierte und Python-ähnliche Skriptsprache, die speziell für die Engine entwickelt wurde.[8] GDScript-Code wird beim Import in Godot in ein internes Bytecode-Format kompiliert und anschlieSSend von der Godot-Laufzeitumgebung ausgeführt.

Umsetzung

Labyrinth

Die Hauptszene besteht grundsätzlich aus nur einer Node, welche das `MazeGenerator`-Skript enthält. Dieses wird automatisch ausgeführt, sobald das Spiel startet und die Szene instanziiert wird. Dabei wird ein Labyrinth als Grid konzipiert, in dem jedes Feld genau $6 \times 6 \times 6$ Meter misst. In einer Schleife wird das Grid vom Ursprung aus und zeilenweise traversiert, sodass in jedem Feld ein vorgefertigter Spielstein erzeugt wird.

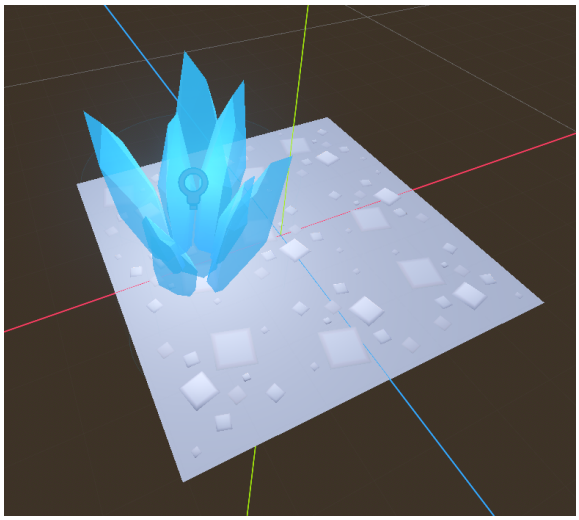


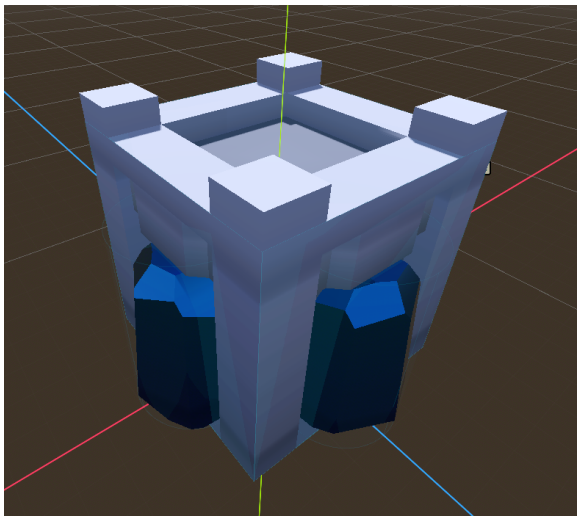
Ob eine Zelle mit einer Wand oder einem Pfad belegt wird, bestimmt der Generator anhand einer einfachen ASCII-Plan-Datei. Diese Datei enthält für jede Zeile des Labyrinths Zeichen, wobei `#` für eine Mauer und ein Leerzeichen für einen Pfad steht. Desweiteren muss es jeweils eine Stelle mit den Sonderzeichen `T` für Target und `S` für Start geben. Auf den beiden reservierten Feldern werden dementsprechend gesonderte Spielsteine mit einem Schatz bzw. dem `PlayerController` platziert.
Für die Mauern und Pfade stehen mehrere Vorlagen zur Verfügung, welche dem Skript in Listen zur Verfügung gestellt werden. Der Generator wählt für jede Zelle per gewichteter

[7] *Reflection probes Godot Engine Documentation (4.4).* URL: `https://docs.godotengine.org/en/4.4/tutorials/3d/reflection_probes.html`.

[8] *GDScript Godot Engine Documentation (4.4).* URL: `https://docs.godotengine.org/en/4.4/getting_started/scripting/gdscript/index.html`.

Zufallsauswahl einen passenden Spielstein aus. Die Mauern sind dabei symmetrisch konzepiert, sodass die Wände zu jeder Seite gleich aussehen und Reflection Probes für jede Seite gesondert gesetzt werden. Die Pfade hingegen sind nicht symmetrisch und beinhalten teilweise auch Gegenstände wie bspw. Tore, die den Weg blockieren könnten. Deshalb überprüft das Skript, dass solche Pfade nicht auf Kreuzungen und ausschlieSSlich entlang eines Korridors korrekt platziert werden können. Die von der Aufgabenstellung geforderten reflektierenden, leuchtenden, transparenten und absorbierenden Oberflächen befinden sich entsprechend diesem Konzept in den Spielsteinen.





Kamera Perspektive

Für die Third-Person Ansicht setzt das Projekt eine Kamerakonfiguration ein, bei der eine Kamera hinter dem Spieler positioniert ist – eine klassische Third-Person Perspektive. Die Kamera wird mit einem Versatz zum Spieler initialisiert. Die Kameraposition wird mit einer linearen Interpolation sanft an die aktuelle Spielerposition angepasst, was Ruckler verhindern soll. Gleichzeitig sorgt ein Raycast vom Spieler zur Kamera dafür, dass die Kamera nicht in Wände gedrückt wird: Erkennt der Raycast eine Kollision, fährt die Kamera automatisch näher heran, bis ein freier Abstand wiederhergestellt ist. Die Blickrichtung der Kamera reagitert zudem auf Mausbewegungen in der Horizontalebene. So kann der Spieler die Kamera um 360° um die eigene Achse schwenken und behält stets das eigene Modell und die unmittelbare Umgebung im Blick. Insbesondere die Feinanpassung von passenden Sensibilitäten hat einige Zeit und Tester beansprucht.