

Projektbericht

“Entwicklung einer KI- und Computergrafik-gestützten Spielumgebung mit Unity”

Gruppenmitglieder

Haitham El Euch
Thorben Meiswinkel
Arman Niaruhi
Peter Kletschka

Betreuer:

Prof. Dr. Volker Blanz
Kathrin Schnieber

WS 2024/2025 - SS 2025

Einleitung

Dieser Projektbericht dokumentiert Konzeption, Umsetzung und Validierung eines First-Person-Dungeon-Spiels. Dieses Spiel basiert auf Unity und verbindet einen prozedural erzeugten Dungeon mit KI-Gegnern, Minispiele und First-Person-Steuerung. Der Spieler bewegt sich auf einer Ebene ohne Sprünge und erkundet eine dynamisch generierte Umgebung. Im Folgenden beschreiben wir Ziele, technische Umsetzung und Teamprozesse.

Technische Umsetzung

Game Engine & Tools

Wir haben das Projekt mit der Game Engine Unity 6 und in C# umgesetzt. Für die Kameraführung kommt Cinemachine zum Einsatz, damit sich Perspektivwechsel und Zielverfolgung ohne eigenen Kameracode sauber steuern lassen. Die Gegner und Minispiele nutzen trainierte ML-Agents Modelle, die als ONNX in Unity geladen und zur Laufzeit ausgeführt werden. Die Inhalte stammen aus eigenen Assets und kuratierten Asset-Paketen, die wir nach Kategorien wie Dungeon, Gegner und Items organisiert haben.

Shader & Materialeffekte

In Unity lassen sich Shader entweder mit dem visuellen Shader Graph erstellen oder direkt in HLSL programmieren. Wir nutzen beide Wege. Shader Graph setzen wir dort ein, wo schnelle Iteration und Parametrisierung wichtig sind. Eigene HLSL-Shader verwenden wir für Effekte mit genauer Kontrolle über Vertex Bewegung und Blending.

Für den Boden existieren zwei Varianten. Die erste Variante bleibt geometrisch ruhig und entsteht im Shader Graph mit dem Voronoi-Node. Aus derselben Graph-Struktur haben wir mehrere Farbschemata abgeleitet, zum Beispiel eine rötliche „Lava“-Fläche, eine „Wasser“-Fläche, eine grünliche Fläche sowie zusätzlich eine neutrale Steinvariante. Technisch ist es eine gemeinsame Shader-Variante mit unterschiedlichen Materialfarben, sodass vier Bodenlooks ohne zusätzliche Meshes verfügbar sind. Die zweite Variante nutzt Vertex Displacement in HLSL. Zwei zeitabhängige Kosinuswellen verschieben die y-Position der Vertices und formen eine weiche Wellenstruktur. Die Fragmentstufe mischt zwei Grundfarben abhängig von der Wellenhöhe und kann optional ein feines UV-Gitter überlagern. Amplitude, Geschwindigkeit, Tiling und Farbwerte sind Materialparameter.

Das Ausgangsobjekt für den Wechsel in die nächste Ebene verwendet einen Shader-Graph mit Emissionsmaske. Eine getaktete Zeitfunktion verschiebt die Maske über die UVs und mischt zwei Farben in den Emissionskanal. So entstehen klare Leuchtstreifen, die den Ausgang deutlich markieren, ohne zusätzliche Geometrie.

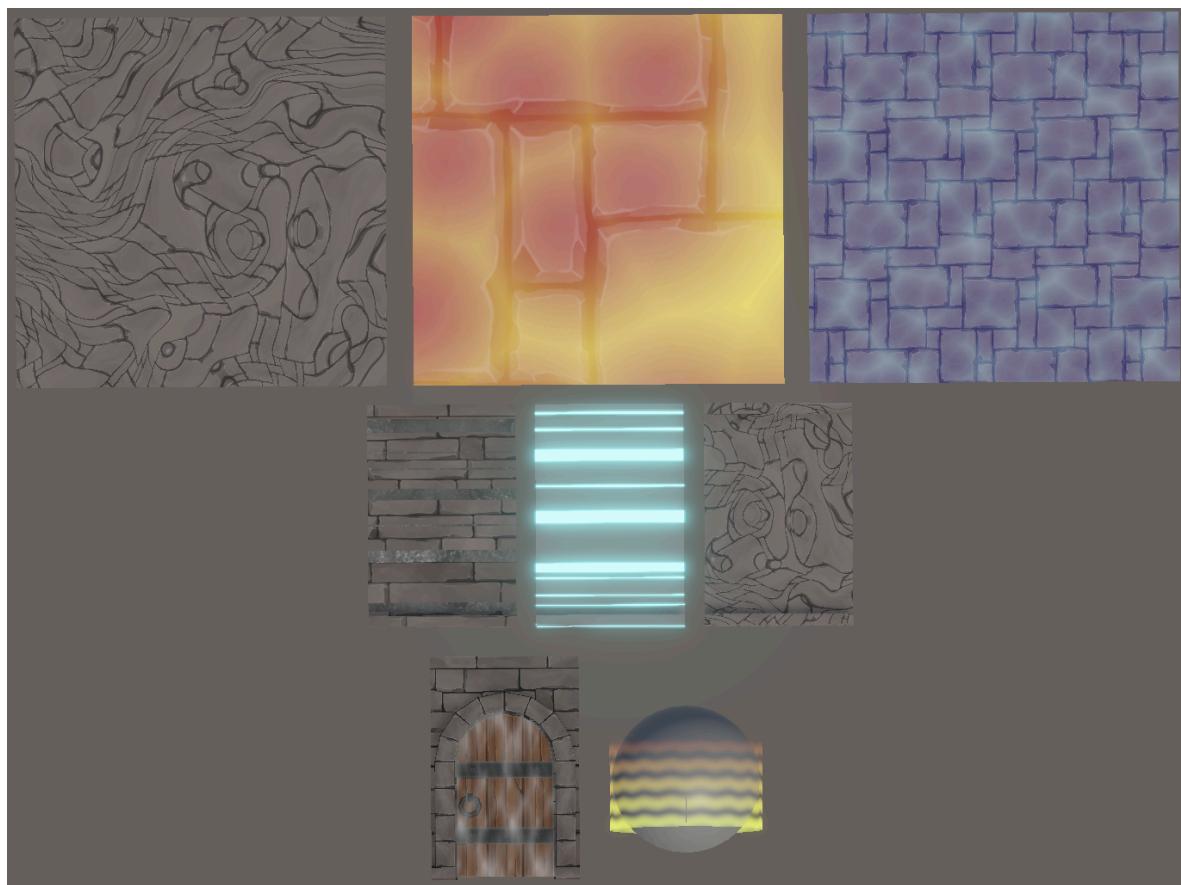
Die brüchige Wand erhält einen Twirl-Graph. Dieser dreht die UVs um ein Zentrum und moduliert die Drehstärke mit Simple Noise. Stärke, Geschwindigkeit und Rauschanteil sind

einstellbar. Das Material wirkt damit leicht verdreht und macht die Instabilität der Wand sichtbar.

Items bekommen einen Aura-Shader. Er rendert additiv ohne Tiefenschreiben, kombiniert einen vertikalen Farbverlauf mit einer zeitabhängigen Welle in den UVs und erzeugt so ein weiches, pulsierendes Leuchten, das Beute zuverlässig hervorhebt.

An den Bosstüren liegt eine Nebelwand. Dafür nutzen wir einen Nebel-Shader mit zwei unterschiedlich skalierten und verschobenen Rausch Layern, die vertikal scrollen. Eine doppelte Pulsfunktion steuert die Emission zwischen einem minimalen und einem maximalen Wert. Der Pass rendert transparent mit Alpha-Blending und ohne Z-Write, dadurch legt sich der Nebel sauber vor die Tür und bleibt halb durchsichtig.

Mehrere Materialeigenschaften werden zur Laufzeit per Skript gesetzt. Der Seed wählt zum Beispiel eine der vier Boden Farbvarianten aus und kann Intensität, Tiling oder Geschwindigkeit der Effekte deterministisch vorgeben. So bleibt ein Durchlauf reproduzierbar und wirkt dennoch abwechslungsreich. Das beigegebene Bild zeigt die Übersichtsszene mit allen verwendeten Shadern.



Architektur (Klassenstruktur, Interfaces)

Das Projekt folgt der komponentenbasierten Architektur von Unity und ist nach Aufgabenbereichen in Ordnern gegliedert. Typische Bereiche sind Dungeon für Generierung und Datenmodell, Manager für Laufzeitsteuerung und Zustände, Controller für Eingabe und Spieleraktionen, Kamera für Kamera Logik, Spawning für das Platzieren von Inhalten, Interaction für Weltinteraktionen, Inventory und Items für Ausrüstung und Beute, Enemy für Gegner, MiniGame für die Zeichenrätsel, Saving für Persistenz, Shooting für Projektilsysteme sowie UI. Jede Komponente besitzt eine klar abgegrenzte Verantwortung und kommuniziert über Datenträger, Interfaces und Ereignisse, statt direkte Abhängigkeiten aufzubauen.

Die Klassenstruktur trennt Daten von Verhalten. Der VoronoiGenerator erzeugt die Geometrie und baut daraus den DungeonGraph mit Räumen, Kantenkennungen und Nachbarschaften. Room, Point, Edge und Triangle sind reine Datenklassen, die keine Spielregeln enthalten. Der GameManagerVoronoi steuert den Ablauf, liest den Seed, startet die Generierung, setzt den Spieler, verfolgt den aktuellen Raum und stößt je nach Raumtyp die passenden Systeme an. Manager wie EventManager, UIManager, CameraManager und ObjectPoolManager kapseln Querschnittsfunktionen. Controller wie FirstPersonPlayerController, PlayerShooting und PlayerPickaxeController setzen Eingaben in Bewegung, Schüsse und Werkzeuge um. Welt Objekte wie Türen und brüchige Wände sind Prefabs mit schlanken Komponenten wie OpenDoor und DestroyableWallInteraction, die ihren Zustand aus dem Save laden und Änderungen wieder zurückschreiben.

Interfaces sorgen dafür, dass Systeme austauschbar bleiben. IInteractable definiert ein einheitliches Verhalten für Objekte, mit denen der Spieler in der Nähe interagieren kann. Das wird bei brüchigen Wänden, Minispieldaten und ähnlichen Interaktionen verwendet und erlaubt Wiederholaufrufe, solange der Spieler im Wirkbereich steht. ISpawnerVoronoi beschreibt das Platzieren von Inhalten in Räumen. Items, Gegner, Minigames und der Boss nutzen jeweils eigene Spawners, die alle dasselbe Interface erfüllen und damit ohne Änderungen am Manager ergänzt oder ausgetauscht werden können. Für die Verteilung von Items kommt ein generischer Distributor zum Einsatz, der Ressourcen bevorzugt ausliefert und sonst nach Seltenheit auswählt. Lose Kopplung entsteht über den EventManager, der Öffnen und Schließen von Türen, die Bossvarianten und die Aktivierung der Zeichenansicht als C#-Events bereitstellt. Komponenten abonnieren die Ereignisse und reagieren, ohne gegenseitig direkte Referenzen zu halten.

Die Abhängigkeitsrichtung bleibt dabei bewusst einfach. Daten fließen vom DungeonGraph zu Spawners, Controllern und UI, Entscheidungen werden im GameManager oder im jeweiligen System getroffen, Rückmeldungen laufen über Events und den Save. Dadurch bleibt die Struktur übersichtlich, Erweiterungen wie neue Gegnertypen, Minispiele oder zusätzliche Raumrollen lassen sich hinzufügen, ohne bestehende Klassen umzuschreiben.

ScriptableObjects & Events

Die ScriptableObjects in Unity sind Objekte von Klassen, die schon vor Programmstart erstellt werden können. Außerdem sind diese Objekte und auch Änderungen an ihnen persistent. Dadurch eignen sie sich gut dazu, Daten zu speichern, sind aber weniger sinnvoll für alle Anwendungen, in denen sich die Werte häufiger ändern. Für die Items haben wir ScriptableObjects für die Daten eines Items genommen, also den Namen etc., damit wir, um ein neues Item in die Spielwelt zu setzen, nicht komplett neue Objekte erzeugen müssen, sondern nur eine neue Referenz auf das Scriptable Objekt. Ein weiterer Vorteil daran ist, dass eine Item Änderung nur in diesem Scriptable Objekt gemacht werden muss und für alle Items dieses Typs gilt.

Der EventManager stellt C#-Events für Türen und die Zeichenansicht bereit. Komponenten wie OpenDoor abonnieren diese Ereignisse und schalten Kollision und Sichtbarkeit ohne direkte Kopplung an andere Skripte. Manager, Gegner und Interaktionen lösen die passenden Ereignisse aus, wodurch Öffnen, Schließen und Boss-Sperren zuverlässig und übersichtlich gesteuert werden.

KI & ML-Agents

Gegner-KI

Für die Entwicklung der Gegner-KI haben wir hauptsächlich **Unity ML-Agents** in Kombination mit **Reinforcement Learning (RL)** eingesetzt. Unity ML-Agents stellt eine Schnittstelle zwischen der Unity-Spielwelt und externen Python-basierten Trainingsskripten bereit und ermöglicht es, KI-Agenten in einer simulierten Umgebung zu trainieren.

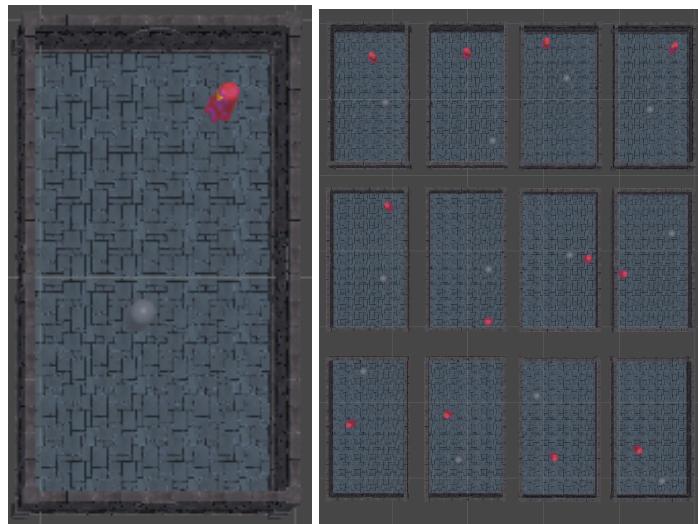
Frameworks und Libraries:

1. **Unity ML-Agents Toolkit:** dient als zentrale Plattform zum Trainieren der Gegner. Unterstützt sowohl kontinuierliche als auch diskrete Aktionen, wodurch flüssige Bewegungen und physikbasierte Drehungen möglich sind. Belohnungssysteme (Rewards) steuern das Verhalten der Agenten. Nach dem Training werden die Modelle im **ONNX-Format** exportiert und in Unity für die Echtzeit-Inferenz.
2. **Python:** Steuert den Trainingsprozess außerhalb von Unity und ermöglicht die Implementierung von Reinforcement-Learning-Algorithmen wie **Proximal Policy Optimization (PPO)**.
3. NumPy und PyTorch: Werden für Datenvorverarbeitung und Netzwerkberechnungen verwendet, insbesondere für komplexere Gegner wie Ghost oder Finder.

Wie wenden wir die Libraries an?

Durch diese Kombination konnten wir verschiedene Gegnerarten realistisch abbilden: Nahkampfgegner wie Bigstalker verfolgen den Spieler flüssig, Fernkampfgegner wie Shooter und Drone nutzen gezielte Projektilangriffe, und Boss-Gegner wie Spawning oder Finder kombinieren trainierte Modelle für komplexe Fähigkeiten wie Spawning oder adaptive Pfadfindung.

Wie haben wir die Gegner trainiert?



Training Umgebung für Bigstalker

Im Grunde haben wir drei verschiedene Fähigkeiten, die je nach Art und Ziel der Fähigkeit mit verschiedenen Parametern konfiguriert werden können. Die Fähigkeiten lauten:

- **Pfadfindung:** Der Gegner kann den Spieler ohne Hindernisse finden.
- **Hindernisumgehung:** Der Gegner umgeht mehrere Hindernisse, um den Spieler zu erreichen.
- **Shooter:** Der Gegner versucht, den Spieler zu erschießen.

Beim Training versucht der Gegner immer, den Spieler zu finden und zu erschießen.

Für das Training versucht der Gegner immer, den Spieler zu finden und zu erschießen. Zu Beginn jeder Episode startet der Spieler an einem zufälligen Ort, und der Gegner handelt entsprechend seiner Aufgabe.

Um das Training effizienter zu gestalten, haben wir ein Prefab erstellt, das sowohl einen Gegner als auch einen Spieler enthält. Durch die Instanziierung mehrerer Prefabs kann ein einzelner Agent gleichzeitig mit vielen verschiedenen Szenarien interagieren. Dadurch kann der Agent die Auswirkungen seiner aktuellen Strategie über mehrere Instanzen hinweg beobachten, was zu effektiverem Feedback, schnellerem Lernen und letztendlich zu einem schnelleren Training führt.

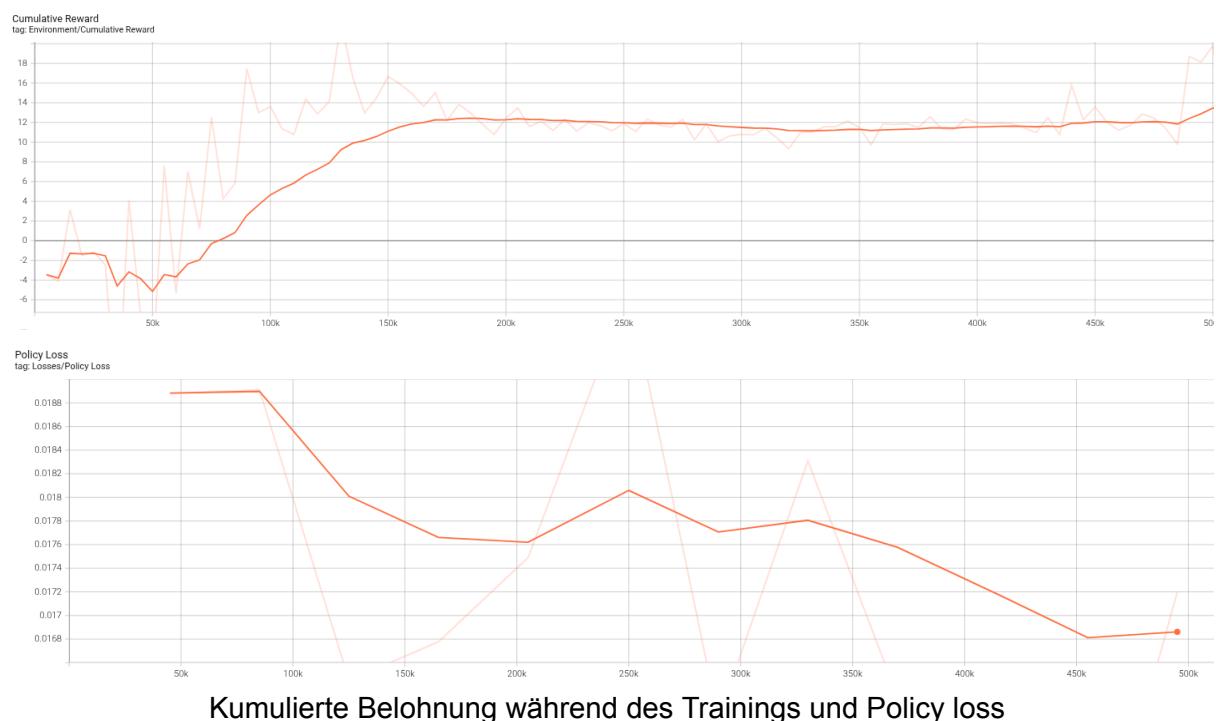
In der Abbildung oben (als Beispiel zeigen wir **Pfadfindung Skill** und **Bigstalker Gegner**) zeigt die linke Seite ein einzelnes Prefab, während die rechte Seite mehrere Prefabs mit Spielern an verschiedenen Orten zeigt. Dank ML-Agents senden alle Prefabs Beobachtungen und Feedback an einen einzigen Agenten, sodass dieser aus mehreren Umgebungen gleichzeitig lernen kann.

Im obigen Bild (das als Beispiele die Fähigkeit **Pfadfindung** und **Bigstalker-Gegner** zeigt) ist auf der linken Seite ein einzelnes Prefab zu sehen, während auf der rechten Seite mehrere Prefabs mit Spielern an verschiedenen Orten zu sehen sind. Dank ML-Agents senden alle Prefabs Beobachtungen und Feedback an einen einzigen Agenten, sodass dieser gleichzeitig aus mehreren Umgebungen lernen kann.

Ergebnisse

Als Beispiel zeigen wir hier das Training des Hunter-Enemy-Agenten: Im Folgenden sieht man **Cumulative Reward** und **Policy Loss** – die den Verlauf des Reinforcement-Learning-Trainings darstellen. Anfangs schwanken die kumulierten Belohnungen stark (Explorationsphase), ab etwa 100.000 Schritten stieg die Performance und stabilisierte sich nach 200.000 Schritten auf einem Plateau mit Werten zwischen 12 und 14. Der Policy Loss sank dabei von ca. 0,0188 auf 0,0168, was den Übergang in eine konvergente Phase verdeutlicht. Der Verlauf des Policy Loss bestätigt diesen Trend.

Zusammenfassend zeigt das Training, dass der Hunter-Enemy-Agent erfolgreich gelernt hat, eine effektive Strategie zu entwickeln, die eine stabile, aber begrenzte Performance ermöglicht.



MiniGames

Für die MiniGames haben wir neuronale Netzwerke zur Klassifizierung von Spielerinteraktionen eingesetzt. Die Spiele „Digit Drawer“ und „Glyph Drawer“ ermöglichen es den Spielern, Zahlen oder Glyphen auf einem Canvas zu zeichnen, die anschließend von der KI erkannt werden.

Frameworks und Libraries:

1. **Unity Sentis / Tensor:** Verarbeitet die gezeichneten Bilder in Echtzeit innerhalb von Unity. Die KI liefert Wahrscheinlichkeiten für alle möglichen Ziffern oder Glyphen und erkennt so die korrekt gezeichneten Symbole.
2. **Python & PyTorch:** Für das Training der Modelle außerhalb von Unity. Es wurden Convolutional Neural Networks (CNN) verwendet, um sowohl Ziffern als auch acht verschiedene Glyphenklassen zuverlässig zu klassifizieren.
3. **ONNX-Format:** Nach dem Training werden die Modelle exportiert und direkt in Unity zur Echtzeit-Inferenz eingebunden.
4. **Matplotlib / Confusion Matrix:** Dient zur Analyse der Klassifikationsleistung während des Trainings (Accuracy, Fehlklassifikationen).
5. **Vortrainiertes MNIST-Modell:** Für die Ziffernerkennung im Digit Drawer haben wir ein bereits vortrainiertes Modell verwendet, um die Klassifizierung zuverlässig und effizient zu gestalten.

Digit Drawer und Details in Implementierung

1. **Fragen-Management (QuestionManager):** Alle Fragen und Antworten sind als Question-Objekte gespeichert. Beim Spielstart wird eine zufällige Frage angezeigt.
2. **Zeichnen auf dem Canvas (CanvasDraw):** Spieler zeichnet die Antwort auf dem Canvas.
3. **KI-Klassifizierung (Classifier):** Die vom Spieler gezeichnete Zahl wird auf 28×28 Pixel skaliert und gegebenenfalls invertiert, bevor sie von der KI über ein neuronales Netzwerk (Unity Sentis / Tensor) verarbeitet wird. Das Modell liefert Wahrscheinlichkeiten für alle möglichen Ziffern, wobei die Ziffer mit der höchsten Wahrscheinlichkeit als Vorhersage verwendet wird.
4. **Überprüfung und Belohnung:** Die vorhergesagte Ziffer wird mit der richtigen Antwort der Frage verglichen (CheckAnswer). Wird die Antwort korrekt erkannt, wird der Zähler für richtige Antworten erhöht und der Fortschritt des Spielers im UI angezeigt, zum Beispiel „3/10 richtig“. Nach Erreichen einer bestimmten Anzahl korrekter Antworten kann ein „The One Ring“ im Spiel gespawnt werden, etwa ein Schlüssel für eine Tür. Bei einer falschen Antwort erhält der Spieler keine Belohnung, kann die Frage aber erneut versuchen.
5. **UI und Spielerfeedback:** Die UI zeigt eine Erfolgsmeldung, zum Beispiel „Done!“ oder „You have The One Ring!“, und das Fragenboard zeigt „No Questions!“.

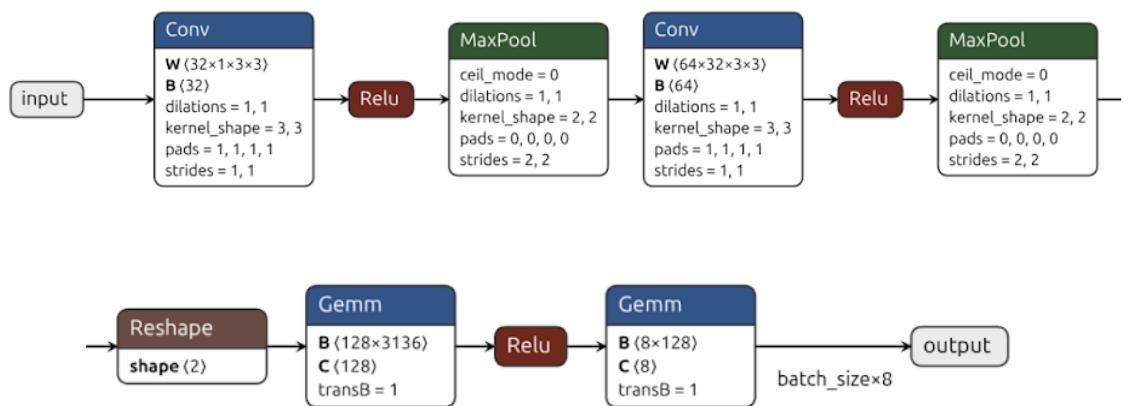
Glyph Drawer und Details in Implementierung

1. **Glyph-Management (CanvasDraw / RefGlyph):** Alle erwarteten Glyphen werden als Liste von Indizes gespeichert, wobei jede Glyphe in einem Dictionary mit einem

Namen verknüpft ist (z. B. 0 → „air“). Das Spiel zeigt dem Spieler den Namen der Glyphe an, die er zeichnen soll. Nach erfolgreichem Zeichnen wird die Glyphe aus der Liste entfernt.

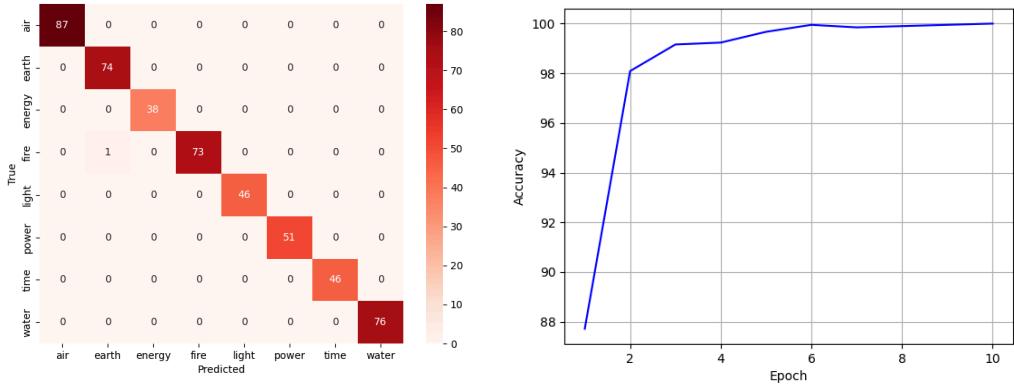
2. KI-Klassifizierung (Classifier):

- Die vom Spieler gezeichnete Glyphe wird auf 28×28 Pixel skaliert, in Graustufen konvertiert und normalisiert. Anschließend verarbeitet die KI das Bild über ein neuronales Netzwerk (Unity Sentis / Tensor), das Wahrscheinlichkeiten für alle möglichen Glyphen ausgibt. Die Glyphe mit der höchsten Wahrscheinlichkeit wird als Vorhersage verwendet.
- Für das Glyphen-Minispiel wurde ein eigenes Dataset mit acht Klassen erstellt: „air“, „earth“, „energy“, „fire“, „light“, „power“, „time“ und „water“ für jede Klasse minimal 250 maximal 500 Beispiele. Die Bilder wurden in Trainings-, Test- und Evaluationssets aufgeteilt (80 %, 10 %, 10 %). Das Convolutional Neural Network (CNN) zur automatischen Klassifizierung besteht aus mehreren Faltungs- und Pooling-Schichten, gefolgt von voll verbundenen Schichten (siehe Architektur in Abb.). Als Loss-Funktion wurde Cross-Entropy Loss verwendet, und der Optimizer war Adam mit einer Lernrate von 0,001.



CNN-Architektur für Glyphennetzwerk verwendet

- Nach Abschluss des Trainings wurde das Modell als ONNX-Datei exportiert und in Unity eingebunden.
- Im Folgenden stellen wir Konfusionsmatrix und Trainingsgenauigkeit pro Epoche bereit. Die Konfusionsmatrix zeigt, dass das Modell die meisten Klassen korrekt vorhersagt und nur wenige Fehlklassifikationen aufweist. Das Diagramm zur Trainingsgenauigkeit pro Epoche zeigt, dass diese schnell ansteigt und sich nach einigen Epochen bei fast 100 % stabilisiert. Zusammen zeigen sie, dass CNN effektiv gelernt hat, die acht Glyphenklassen mit hoher Genauigkeit zu klassifizieren.



Confusion Matrix und auch Accuracy von Evaluation

3. **UI und Spielerfeedback:** Die aktuelle Glyphe, die der Spieler zeichnen soll, wird deutlich im UI angezeigt, zum Beispiel „fire“. Während der Spieler die Glyphe auf dem Canvas zeichnet, verarbeitet die KI das Bild in Echtzeit und zeigt Wahrscheinlichkeiten für mögliche Vorhersagen an, sodass der Spieler direkt sehen kann, wie gut seine Zeichnung erkannt wird. Gleichzeitig wird der Fortschritt der Spieleraufgaben angezeigt, sodass er nachvollziehen kann, welche Glyphen bereits korrekt gezeichnet wurden. Sobald alle Glyphen richtig gezeichnet sind, wird ein spezielles Objekt, der Boss-Schlüssel, in der Nähe des Canvas in der Spielwelt gespawnt. Die UI informiert den Spieler über diesen Erfolg mit einer Nachricht wie „You have the boss key!“. Wenn keine Glyphen mehr zur Verfügung stehen, wird die Meldung „No Glyphs!“ angezeigt, um dem Spieler klarzumachen, dass alle Aufgaben abgeschlossen sind.

Versionierung & Zusammenarbeit

Wir arbeiten mit einem GitHub Repository. Aufgaben werden in einem Kanban Board gepflegt mit Spalten für Backlog, in Arbeit, Review und erledigt. Änderungen entstehen auf kurzen Feature Branches und werden per Pull Request in den Hauptzweig gemerkt. Im Review prüfen wir Code, Szenenreferenzen und Prefab Änderungen. Für die Abstimmung nutzen wir Discord für Meetings und Screensharing. Eine WhatsApp Gruppe dient für kurze Absprachen und Erinnerungen. Feste Seeds und Debug Einblendungen helfen uns, Fehler reproduzierbar zu machen und denselben Levelstand gemeinsam nachzustellen.

Build-Prozess

Wir erstellen Builds für Windows, Linux und macOS. Die Szenenreihenfolge wird in den Build Settings gepflegt, alle benötigten Prefabs, Assets und ONNX-Modelle sind im Projekt referenziert. Vor dem Export wählen wir die Zielplattform und erzeugen das Paket direkt aus Unity, sodass die erzeugten Builds ohne zusätzliche Nacharbeit lauffähig sind.

Lösungsansätze

Dungeon-Generierung

Da wir einen Dungeon mit unterschiedlich geformten Räumen haben wollten, haben wir uns dafür entschieden, den Dungeon zufällig zu generieren. Das Problem daran war, dass es sehr komplex gewesen wäre, den Dungeon zufällig mit verschiedenen großen und geformten Räumen zu erstellen, weil die Räume ineinander hätten generiert werden können. Um dieses Problem zu lösen, entschieden wir uns, den Dungeon als Voronoi-Diagramm zufälliger Punkte innerhalb eines Quadrats zu generieren. Die Größe des Quadrats gibt die Dungeon-Größe an und ist wie die Anzahl der Voronoi-Sites konfigurierbar.

Die Implementierung des Voronoi-Diagramms findet mit Hilfe der Delaunay-Triangulierung statt, da zwischen dem Voronoi-Diagramm und der Triangulierung eine Dualität besteht. Die Delaunay-Triangulierung ist über den Bowyer-Watson-Algorithmus implementiert.

Geometrie

Um die Implementation der Algorithmen zu vereinfachen, sind vier Helfer Klassen geschrieben worden. Diese sind:

1. Point
2. Edge
3. Triangle
4. Circle

Point

Die Point Klasse enthält die Koordinaten des Punktes als zwei Floats, eine Methode zum Vergleich zweier Punkte und eine Methode, um die Distanz zwischen zwei Punkten zu berechnen.

Edge

Die Edge-Klasse wird benutzt, um eine Kante anhand von zwei Punkten darzustellen. Neben dem Start- und Endpunkt bekommt jede Edge eine eigene ID, die später bei der Erstellung des Graphen und der Platzierung der Türen verwendet wird. Außerdem gibt es auch hier eine Vergleichsmethode, aber auch eine Methode, die prüft, ob zwei Edges sich schneiden.

Triangle

Das Dreieck wird als Liste von drei Punkten erzeugt, enthält aber auch eine Liste der drei Kanten. Außerdem kann bei einem Dreieck geprüft werden, ob ein Punkt oder eine Kante in dem Dreieck ist, sowie ob sich ein Punkt im Umkreis des Dreiecks befindet.

Circle

Diese Klasse speichert einen Kreis durch einen Mittelpunkt und einen Radius und ist eine reine Datenklasse.

Bowyer-Watson

Der Bowyer-Watson-Algorithmus generiert die Delaunay-Triangulierung, indem zunächst ein Super Dreieck generiert wird. Dieses Super Dreieck ist groß genug, dass alle anderen Punkte innerhalb dieses Dreiecks liegen. Danach passieren für jeden Punkt nacheinander die folgenden drei Schritte:

1. Füge den Punkt hinzu
2. Prüfe welche Dreiecke der Triangulierung verletzt wurden (ein Dreieck gilt als verletzt, wenn ein Punkt der nicht zum Dreieck gehört innerhalb des Umkreises liegt)
3. Entferne die verletzten Dreiecke und füge neue hinzu, die eine gültige Triangulierung liefern

Sind alle Punkte hinzugefügt worden, werden alle Dreiecke, die einen Punkt des Super Dreiecks beinhalten, entfernt. Dadurch erhalten wir am Ende eine Liste von Dreiecken, die eine Delaunay-Triangulierung ergeben.

Voronoi-Diagramm

Um aus der Liste von Dreiecken ein Voronoi-Diagramm zu erstellen, müssen die Umkreismittelpunkte der benachbarten Dreiecke verbunden werden. Falls ein Dreieck keine drei Nachbarn hat (also am Rand der Triangulierung liegt), werden die Mittelsenkrechten jeder Seite ohne Nachbarn gebildet und gehen bis in die Unendlichkeit. Da unser Dungeon aber innerhalb einer Begrenzung liegt, müssen diese Kanten nur bis an die Dungeon Grenze verlängert werden und nicht bis in die Unendlichkeit. Allerdings war es nicht sehr einfach herauszufinden, welche Dreiecke Nachbarn sind, weshalb wir den Algorithmus angepasst haben (mehr zu den Problemen im Kapitel Probleme & Lösungen).

Wir haben den Algorithmus wie folgt angepasst:

1. Bilde die Mittelsenkrechten für jede Seite in jedem Dreieck
2. Gehe durch alle Mittelsenkrechten und prüfe, ob sie einen gemeinsamen Punkt haben (da die Edges alle so definiert wurden, dass der Umkriesmittelpunkt der Startpunkt und der Mittelpunkt der Dreieckskante der Endpunkt sind, ist das relativ simpel)
3. Verbinde die beiden Mittelsenkrechten zu einer längeren Kante
4. Verlängere alle Mittelsenkrechten, die nicht mit einer anderen verbunden wurden, bis zum Rand des Dungeons

Am Ende haben wir eine Liste von Edges, aus denen wir den Dungeon zusammenbauen.

Dungeon in die Welt setzen

Die Liste der edges ist bisher nur ein Code-Objekt und noch nichts in der Spielwelt sichtbares. Um den Dungeon nun bespielbar zu machen, haben wir eine Methode geschrieben, die entlang einer Edge Mauern platziert. Diese Methode berechnet als erstes die Länge der Kante, daraus wird berechnet, wie das Mauer Bauteil ("Prefab") skaliert werden muss und wie viele Mauern benötigt werden. Denn wir setzen nicht eine lange Mauer, sondern mehrere Segmente, um die Platzierung der Türen und zerstörbaren Wände zu vereinfachen. Diese Methode rufen wir für jede Edge in unserem Voronoi-Diagramm auf. Zusätzlich wird an jedem Start- und Endpunkt eine Säule generiert, damit der Schnittpunkt der verschiedenen Mauern überdeckt wird.

Dungeon-Graph

Die Voronoi-Schritte liefern die Geometrie der Räume, also Mittelpunkte und Kanten. Damit daraus ein spielbares Level mit Regeln und Zuständen wird, legen wir einen Dungeon-Graphen an. Jeder Raum wird als Knoten mit stabiler ID und Mittelpunkt gespeichert. Zwei Knoten gelten als Nachbarn, wenn ihre Zellen im Voronoi-Diagramm eine gemeinsame Kante besitzen. Diese Kante erhält eine eindeutige Kanten-ID. Über diese IDs finden wir später die zugehörigen Tür- oder Wandobjekte in der Szene wieder und können sie gezielt ansteuern.

Zu jedem Raum halten wir zusätzliche Daten fest. Der Typ beschreibt die Rolle im Spielablauf. Der Besuchsstatus erlaubt es, Fortschritt auf der Karte zu markieren und Wiederholungen zu vermeiden. Der Innenradius ist der Radius des größten einpassenden Kreises im Raum. Er dient als sicherer Platzierungsbereich für Gegner, Items, Minispiele und Hindernisse. Für schnelle Zugriffe führen wir Lookup-Tabellen von ID zu Raum, von Mittelpunkt zu Raum und von Kanten-ID zu Tür-Objekt. Die Nachbarschaften leiten wir aus der Delaunay-Triangulierung ab, da dort stabile Nachbarbeziehungen entstehen.

Vor dem Einsatz im Spiel filtern wir sehr kurze Voronoi-Kanten heraus. Auf solchen Minikanten könnten Türen nur als enge Spalten entstehen, was zu Blockaden, Clipping oder unklaren Situationen führen würde. Durch das Filtern bleiben die Durchgänge breit genug, die Ausrichtung sauber und die Lesbarkeit hoch. Für Wegberechnungen verwenden wir eine Breitensuche. Auf Wunsch schließen wir den Bossraum aus, damit Pfade zu frühen Zielen nicht durch den Bossbereich führen. Diese Wahl verhindert Sackgassen und hält den Spielfluss klar.

Raumtypen und Spielfortschritt

Nach dem Aufbau des Graphen weisen wir den Räumen Rollen zu. Der Startraum wird zufällig aus allen Räumen gewählt und dient als Spawnpunkt. Der Bossraum ist der Raum, der vom Start am weitesten entfernt liegt. Diese Wahl sorgt dafür, dass der Boss nicht direkt neben dem Start liegt und der Spieler erst einige Entscheidungen trifft, bevor er der Arena begegnet. Ein konfigurierter Anteil der übrigen Räume wird zu Itemräumen, Gegnerräumen und Minigameräumen. Der Rest bleibt normale Räume, die nur als Verbindungen dienen.

Damit Fortschritt zuverlässig möglich bleibt, bestimmen wir einen kürzesten Pfad vom Start zu einem Itemraum. Wenn möglich, vermeiden wir dabei den Bossraum. Die Kanten dieses Pfads markieren wir als Pflichtkandidaten für Türen. Damit stellen wir sicher, dass der Spieler mindestens eine klare Route zu einer frühen Belohnung wie der Spitzhacke hat. So ist garantiert, dass der Spieler die nötige Ausrüstung erhält und das Level nicht in sich geschlossen bleibt.

Türen

Wände entstehen segmentiert entlang der Voronoi-Kanten. Nur das mittlere Segment einer ausreichend langen Kante ist als Tür geeignet. Diese Regel verhindert, dass Türen zu nah an Ecken sitzen oder schräg geschnitten werden. Ob an einem Kandidaten eine Tür erscheint, bestimmen drei Bedingungen. Liegt die Kante an der Grenze zum Bossraum, wird dort eine Bosstür platziert. Liegt die Kante auf dem garantierten Pfad vom Start zu einem Itemraum, entsteht dort sicher eine Tür. Auf allen anderen Kandidaten entscheidet eine Grundwahrscheinlichkeit. So bleibt die Struktur nachvollziehbar und gleichzeitig abwechslungsreich.

Bosstüren erhalten eine besondere Behandlung. Wir richten das Eingangsobjekt zum Bossraum aus und korrigieren die Orientierung, falls nötig. Eine Nebelwand wird aktiviert und das sichtbare Türpanel schmäler eingestellt, damit die Passage erkennbar, aber gesperrt ist. Jede Tür registrieren wir unter ihrer Kanten-ID im Graphen. Ein Skript hört auf Ereignisse des Event-Managers und schaltet beim Öffnen den Collider ab und blendet das Tür-Objekt aus. Beim Schließen passiert das Gegenteil. Für Bosstüren gibt es eigene Ereignisse, damit sie unabhängig von normalen Türen geöffnet und geschlossen werden können. Diese Entkopplung verhindert, dass ein Raumkampf versehentlich eine Bosstür beeinflusst.

Zerstörbare Wände

Wenn an einem Kandidaten keine Tür entsteht und keine Bosstür nötig ist, setzen wir dort häufig eine zerstörbare Wand. Diese Elemente sind Abkürzungen oder Geheimwege. Jede Wand besitzt wenige Lebenspunkte und lässt sich nur mit der Spitzhacke abbauen. Beim Treffer zeigen Textanzeigen kurz die aktuelle Lebenspunkte in einer passenden Farbe. Jede Wand erhält einen festen Index, über den Aktivzustand und Lebenspunkte gespeichert werden. Bei null Lebenspunkten wird das Objekt deaktiviert und bleibt auch nach einem Laden zerstört. Die Interaktion prüft, ob die Spitzhacke in der rechten Hand ausgerüstet ist, blendet bei Bedarf kurze Hinweise ein und sperrt während des Schwungs Bewegung und Schießen, damit die Rückmeldung eindeutig bleibt.

Platzierung

Für alle Platzierungen nutzen wir den Innenradius des Raums als Sicherheitszone. Dadurch stehen Objekte nicht an Kanten, schneiden nicht in Türen und bleiben gut erreichbar. Die Verteilung erfolgt kreisförmig um den Mittelpunkt. Wir verwenden einen festen Abstand vom Zentrum, der sich an einem Anteil des Innenradius orientiert und zusätzlich gedeckelt ist. So entstehen gleichmäßige Muster, die in kleinen wie in großen Räumen funktionieren.

In Itemräumen erscheinen zwischen einer und vier Belohnungen. Ein Verteiler wählt die Gegenstände nach hinterlegter Seltenheit. Muss-Gegenstände wie Glyphen oder die Spitzhacke werden vorgezogen. Reicht die geplante Zahl an Slots nicht aus, erhöhen wir die Zuweisung über die Räume, bis alle Pflichtgegenstände im Level liegen. Jedes Item bekommt einen globalen Index, damit der Einsammelstatus zuverlässig gespeichert und

wiederhergestellt werden kann. Auren und Effekte können sich an der Seltenheit orientieren und werden deterministisch gesetzt.

Gegner entstehen in Gegneräumen in Abhängigkeit von der Raumgröße. Sie werden kreisförmig um den Mittelpunkt platziert und bleiben zunächst inaktiv. Beim Betreten des Raums werden sie aktiviert und die Türen schließen. Jeder Tod reduziert einen Zähler. Spawnt ein Gegner Helfer, erhöht sich der Zähler wieder. Sobald der Zähler null erreicht, öffnen sich die Türen automatisch. Werte und Lebenspunkte werden pro Ebene skaliert, damit der Schwierigkeitsgrad mit dem Fortschritt wächst.

Im Bossraum platzieren wir zunächst Hindernisse innerhalb des Innenradius. Diese Objekte strukturieren den Raum, bieten Deckung und verhindern, dass Kämpfe im leeren Feld stattfinden. Der Boss steht je nach Konfiguration zentral oder verteilt sich bei mehreren Entitäten auf einen inneren Bereich. Beim Betreten schließen die Bosstüren. Nach dem letzten Treffer öffnen sie wieder. Flags für Bosssieg und geöffnete Bosstüren werden gesetzt und ein Ausgang erscheint an einer festen Position im Raum. Ist der Boss im Speicherstand bereits besiegt, wird beim Laden sofort der Ausgang erzeugt und die Türen bleiben offen. So bleibt die Ebene konsistent und der Spieler kommt ohne erneuten Kampf weiter.

In Minispielräumen erscheint ein Minispiel-Prefab am Raumzentrum. Die Auswahl ist über den Seed reproduzierbar. Die Drehung ist zufällig, die Höhe fest, damit die Interaktionszonen zuverlässig erreichbar sind. Hinweise erscheinen in der Nähe und verschwinden wieder, wenn der Spieler den Bereich verlässt.

GameManager

Der GameManager liest den Seed und erzeugt daraus die Voronoi-Struktur. Anschließend baut er den Dungeon-Graphen, weist die Raumtypen zu, spawnt den Spieler und stellt den Speicherzustand wieder her. Dazu gehören besuchte Räume, die aktuelle Raum-ID, Türzustände im Bossbereich, aktive und zerstörte Wände, einsammelbare Items sowie die Ausrichtung von Spieler und Kamera.

Während des Spiels ermittelt der Manager den aktuellen Raum über die Distanz zum Mittelpunkt. Wechselt der Spieler in einen Nachbarraum, löst der Manager die passende Logik aus. In Gegneräumen werden Gegner aktiviert und die Türen geschlossen. Im Bossraum wird der Boss aktiviert und die Bosstüren schließen. In Minigameräumen stehen die Interaktionen sofort bereit.

Türen reagieren auf ereignisgesteuerte Aufrufe, wobei normale Türen und Bosstüren getrennt behandelt werden. Spawner für Items, Gegner, Boss und Minigames werden in einer festen Reihenfolge initialisiert. Dadurch funktioniert das Fortsetzen nach einem Laden ohne Nebeneffekte und ohne fehlende Referenzen.

Speichern und Seed

Der gleiche Seed erzeugt die gleiche Punkteverteilung und damit die gleiche Voronoi-Struktur. Raum-IDs und Kanten-IDs bleiben stabil und bilden die Grundlage für die Speicherung. Besuchsstatus, aktuelle Raum-ID, Bosszustände, zerstörbare Wände mit Lebenspunkten, Itemaktivität sowie Position und Blickrichtung des Spielers werden gesichert und beim Laden wiederhergestellt.

Dekorationen wie Gras, Bäume und Felsen nutzen vom Hauptseed abgeleitete Salts. So sind sie reproduzierbar, ohne andere Kategorien zu beeinflussen. Materialien, Licht und Skybox werden ebenfalls deterministisch gewählt, damit ein Durchlauf konsistent bleibt.

Implementierungsdetails

Voronoi und Delaunay ergänzen sich. Die Delaunay-Dreiecke liefern stabile Nachbarschaften, die Voronoi-Kanten definieren klare Raumgrenzen. Eine Türplatzierung in der Mitte eines Wandsegments sorgt für gleichmäßige Durchgänge und vermeidet doppelte Öffnungen an Knotenpunkten. Eine Mindestlänge für Türkandidaten verhindert zu enge Öffnungen, die sonst zu Kollisionen führen könnten.

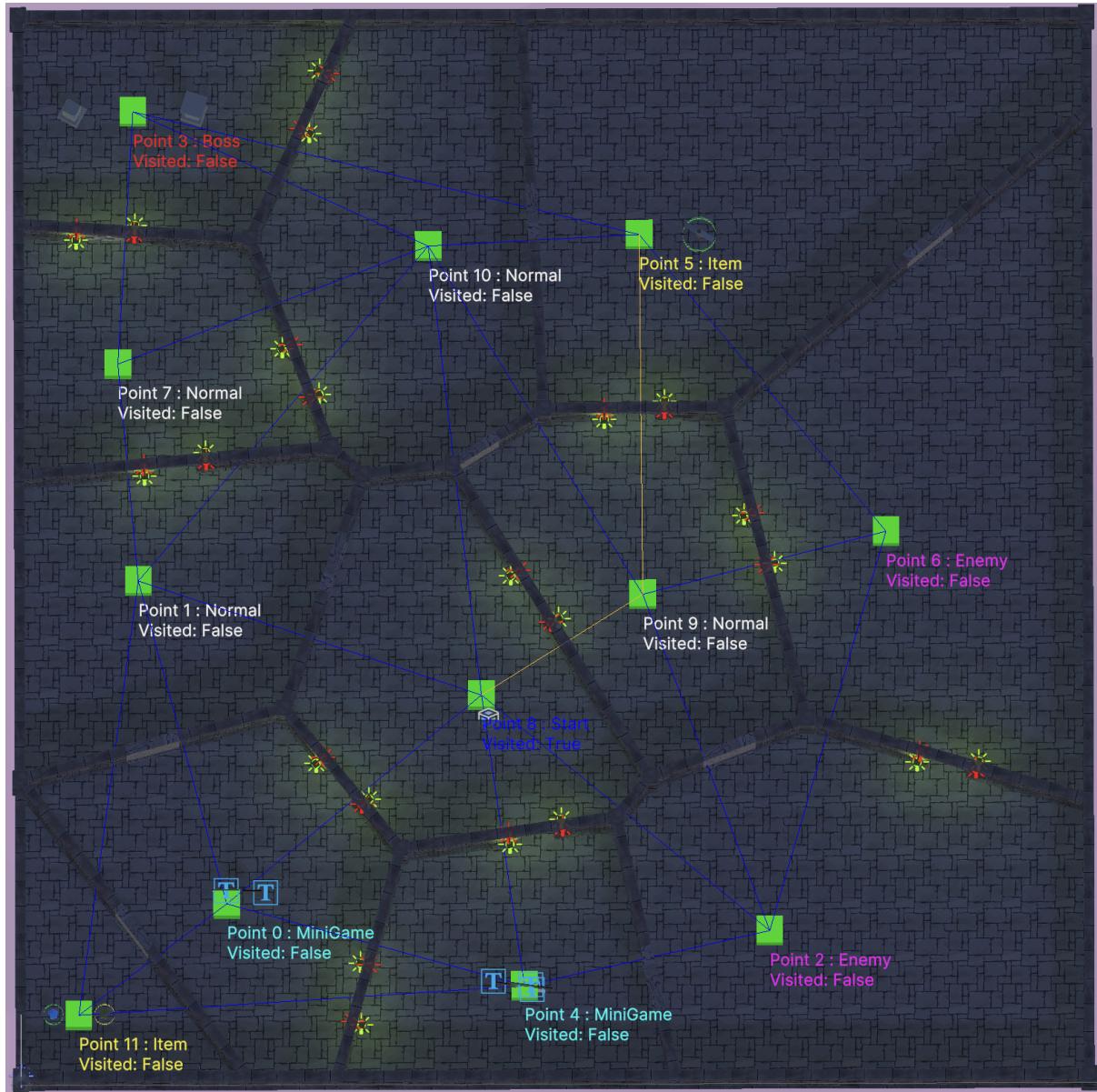
Der garantierte Pfad vom Start zu einem Itemraum stellt sicher, dass frühe Ziele erreichbar sind. Zerstörbare Wände verändern Wege nur optional und bleiben durch den Speicherstand konsistent. Die Ereignissteuerung der Türen reduziert die Logik an den Objekten selbst auf das Umschalten von Collider und Sichtbarkeit.

Insgesamt bleibt die Umsetzung effizient und sorgt für eine nachvollziehbare, stabile Struktur des Dungeon-Graphs.

Debug

Für Entwicklung und Tests können wir Zufallspunkte, Delaunay-Dreiecke, Voronoi-Kanten, Mittelsenkrechten, Zellzentren und den ermittelten Itempfad sichtbar machen. Kanten-IDs werden im Raum angezeigt, Türen tragen ihre Kanten-ID, und reine Wände sind entsprechend markiert. Für den Bossraum gibt es eine Ausgabe der Nachbarn und der verwendeten Türkanten.

Zur Visualisierung nutzen wir Gizmos, um Platzierungen und Logik direkt im Editor überprüfen zu können. So lässt sich leicht kontrollieren, ob die Generierung korrekt arbeitet und ob der Voronoi-Generator erwartungsgemäß funktioniert. Diese Ansichten helfen, Platzierung und Türlogik zu prüfen und Randfälle schnell zu erkennen.



Karte

Die Karte greift direkt auf den Dungeon-Graphen zu. Räume werden als Knoten mit ihrem jeweiligen Typ dargestellt, Verbindungen erscheinen als Linien. Besuchte Räume sind von unbesuchten zu unterscheiden, und der aktuelle Raum ist deutlich erkennbar. Die Blickrichtung des Spielers kann als Pfeil markiert werden.

Geöffnete Bosstüren werden als Zustand geführt. Da die Karte als Einblendung über dem laufenden Spiel liegt, bleibt die Welt aktiv und die Orientierung für den Spieler trotzdem klar.

Tuning und Validierung

Dungeongröße und Anzahl der Voronoi-Punkte wachsen mit der Ebene. Mindestlänge für Türkandidaten, Grundwahrscheinlichkeit für optionale Türen, Sicherheitsabstände für Platzierungen sowie Mengen für Items, Gegner, Hindernisse und Deko sind konfigurierbar. Farben, Licht und Skybox werden aus dem Seed abgeleitet. Für Deko verwenden wir vom Hauptseed abgeleitete Salts, damit die Kategorien unabhängig voneinander reproduzierbar bleiben. Werte von Gegnern und Bossen werden pro Ebene skaliert, sodass der Schwierigkeitsgrad nachvollziehbar anzieht.

Die Validierung erfolgt zurzeit manuell und laufzeitnah. Dafür gibt es umfangreiche Sichtbarkeiten im Editor. Punkte, Delaunay-Dreiecke, Voronoi-Kanten, Mittelsenkrechten, Zellzentren und der berechnete Pfad vom Start zu einem Itemraum können eingeblendet werden. Türen tragen ihre Kanten-ID, reine Wände sind klar markiert. Für den Bossraum gibt es eine Ausgabe mit Nachbarn und verwendeten Türkanten. Mit festen Seeds lassen sich Konstellationen exakt nachstellen. So prüfen wir, ob der garantierte Pfad vom Start zu einem Itemraum tatsächlich begehbar ist, ob Türen nach Kämpfen und nach dem Laden korrekt öffnen und schließen, ob Besuchsstatus, aktuelle Raum-ID, zerstörbare Wände samt Lebenspunkten und Itemaktivität im Speicherstand stabil sind und ob der Wechsel der Räume zuverlässig erkannt wird. Zusätzlich testen wir bewusst dünne Zellen und sehr kurze Kanten, damit keine unbrauchbaren Durchgänge entstehen, sowie Gegner mit Helfern, damit der Türzähler erst bei null wirklich öffnet. Für den Bossraum prüfen wir Ausrichtung des Eingangs, Schließen beim Betreten, Öffnen nach dem Sieg und das Erscheinen des Levelausgangs.

Insgesamt entsteht aus der Voronoi-Geometrie eine nachvollziehbare Levelstruktur. Der Dungeon-Graph verknüpft Räume, Türen und Wände mit klaren Regeln, der Innenradius hält Platzierungen sauber, und der GameManager verbindet Generierung, Spawner, Ereignisse und Speicherung zu einem konsistenten Ablauf, der sich reproduzieren lässt und trotzdem bei jedem Durchlauf frisch wirkt.

Gegner-KI und Kampfmechaniken

Wir haben im Rahmen der Entwicklung insgesamt 8 Gegnertypen konzipiert, nämlich Nahkampf-, Fernkampf- und Kombinationsgegner als Boss-Gegner. Diese wurden einerseits mithilfe der ML Agents-Bibliothek und verstärktem Lernen trainiert, wie wir in den folgenden Kapiteln erläutern werden, und andererseits haben wir bereits trainierte Modelle kombiniert, um neue Gegnertypen mit kombinierten Fähigkeiten zu entwickeln.

Die Enemy-Typen sind:

1. Nahkampf-Gegner
 1. Bigstalker
 2. Ministalker
 3. Ghost
2. Fernkampf-Gegner
 1. Shooter
 2. Drone

3. Boss-Gegner
1. Spawner 1 (kleine Gegner)
2. Spawner 2 (Dronen)
3. Finder

Im Folgenden werden wir jedem Typ näher erläutern:

Nahkampf-Gegner



Bigstalker

Der „Bigstalker“ ist ein Nahkampfgegner, der den Spieler aktiv verfolgt. Seine Bewegungssteuerung erfolgt über ein neuronales Netzwerk mit kontinuierlichen Aktionsaufgaben, wodurch flüssige, physikbasierte Bewegungen und Drehungen ermöglicht werden. Zu den Eingabedaten des Netzwerks gehören der Richtungsvektor zum Spieler (3 Werte), die eigene Vorwärtsrichtung (3 Werte) und der Winkel zum Spieler (1 Wert). Das Netzwerk generiert zwei kontinuierliche Aktionswerte als Ausgaben, die die Vorwärts-/Rückwärtsbewegung und die Drehung nach links oder rechts steuern. Das Belohnungssystem ist so konstruiert, dass es positive Belohnungen für das Annähern an den Spieler, das Ausrichten auf den Spieler und das Erreichen des Spielers gibt. Negative Belohnungen gibt es für Kollisionen mit Wänden, das Steckenbleiben oder ineffizientes Verhalten. Typischerweise ist der Bigstalker ein einzelner, größerer und stärkerer Gegner.

Ministalker

Diese Art von Gegner verhält sich genauso wie Bigstalker, mit dem entscheidenden Unterschied, dass sie kleiner und schneller sind und in größerer Zahl vorkommen.

Ghost

Der „Ghost“ ist ein agiler Nahkampfgegner, der sich unvorhersehbar verhält. Seine Bewegungen werden durch ein neuronales Netzwerk gesteuert. Das Netzwerk verwendet den normalisierten Richtungsvektor zum Spieler, seine eigene Vorwärtsrichtung und den normalisierten Winkel zum Spieler als Eingabedaten, insgesamt sieben Werte. Der Ghost besitzt besondere Spezialfähigkeiten, die seine Interaktion mit dem Spieler beeinflussen. Dazu gehören die Fähigkeit, sich selbst unsichtbar zu machen, wobei die Durchsichtigkeit des Materials regelmäßig verändert wird, um die visuelle Wahrnehmung zu beeinträchtigen,

sowie eine Dashbewegung, bei der eine schnelle seitliche Bewegung mittels linearer Interpolation (Lerp) ohne Teleportation erfolgt. Das Belohnungssystem des Netzwerks ist so konzipiert, dass es positive Belohnungen für die Reduzierung der Entfernung zum Spieler, die Annäherung an den Spieler und die Körperberührungen des Spielers gibt. Negative Belohnungen gibt es bei Kollisionen mit anderen Hürden, beim Feststecken oder bei ineffizienter Zeitnutzung.

Fernkampf-Gegner

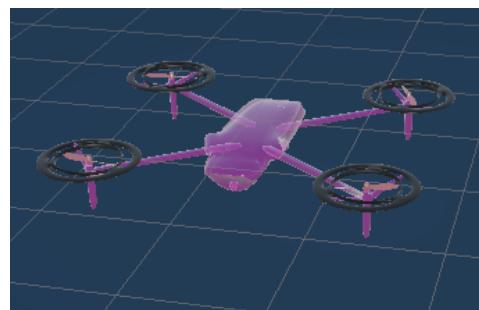
Shooter



Der „Shooter“ ist ein unbeweglicher und stationärer Gegner, dessen Bewegungsmöglichkeiten durch einen Rigid Body eingeschränkt sind und der den Spieler mit gezielten Projektilangriffen bekämpft. Zu seinen Steuerungsmöglichkeiten gehört die horizontale Drehung, um sich auf den Spieler zu orientieren. Das Netzwerk verwendet den Richtungsvektor zum Spieler, das Leben des Spielers und den Winkel zum Ziel als Eingaben. Die Aktionen umfassen sowohl kontinuierliche Drehbewegungen als auch eine diskrete Schlussaktion.

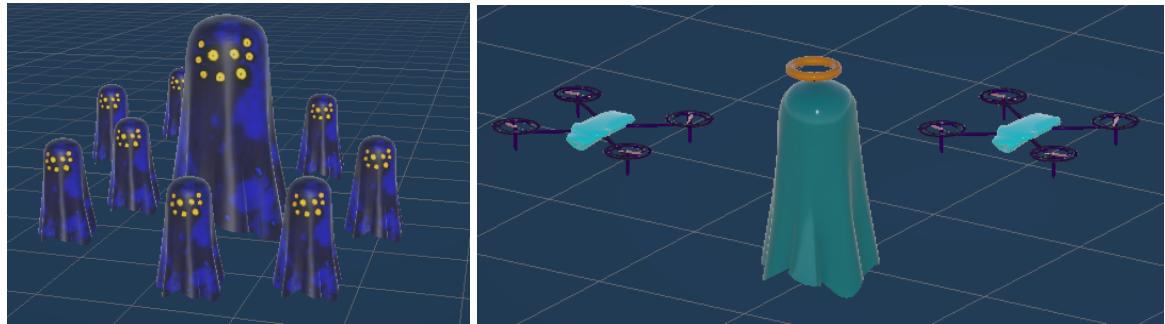
Die Projektilmechanik nutzt Object-Pooling, um die Berechnungseffizienz zu erhöhen und eine Abkühlzeit von 0,5 Sekunden zwischen den Schüssen zu implementieren. Die Genauigkeit der Angriffe wird durch die exakte Ausrichtung des Schusspunkts bestimmt. Das Belohnungssystem ist so konzipiert, dass präzises Zielen und das Besiegen des Spielers positiv verstärkt werden, während ineffiziente Aktionen, wie z. B. Zeitverschwendungen, negativ bewertet werden.

Drone



Eine weitere Variation des Shooters ist die „Drohne“, ein fliegender Gegner, der nach denselben Methoden wie der Shooter operiert, jedoch zusätzlich in der Lage ist, in großer Höhe zu agieren. Die Drohne kann ihre Position im dreidimensionalen Raum kontinuierlich anpassen und den Spieler mit gezielten Projektilangriffen angreifen.

Boss-Gegner



Spawnlings

Der Spawnling ist ein beweglicher Nahkampfgegner, der den Spieler aktiv verfolgt und außerdem die Fähigkeit besitzt, kleinere Gegner zu spawnen. In Bezug auf seine Bewegungsfähigkeiten ähnelt er Gegnern wie dem Big Stalker und dem Small Stalker. Eine der besonderen Fähigkeiten des Spawnling ist es, zusätzliche kleine Spawnlinge zu spawnen, um den Spieler zu überwältigen. Außerdem passt er seine Bewegungsgeschwindigkeit dynamisch an die Entfernung zum Spieler an. Sowohl die Spawn-Rate als auch die Art der erzeugten Gegner können mithilfe vordefinierter Prefabs konfiguriert werden. Die Bilder zeigen zwei Arten von Prefabs, die wir miteinander verbunden haben: Drones und Smallstalkers.

Finder

Der „Finder“ ist ein Verfolger-Gegner, der den optimalen Pfad zum Spieler berechnet, selbst in komplexen Umgebungen. Seine Steuerung umfasst dynamische Hindernisvermeidung sowie adaptive Pfadfindung, die es ihm ermöglicht, Wände und Türen effizient zu umgehen. Das Verhalten des Finders ist eingeschränkt, wenn der Spieler für ihn unzugänglich ist, wodurch realistische Einschränkungen in der Bewegung und Entscheidungsfindung simuliert werden.

Die folgende Tabelle bietet einen Überblick über die Eigenschaften der verschiedenen Gegnertypen und des Spielers:

Type	Leben	Schaden	Geschwindigkeit	Spawn-Intervall
Player	100	5	7	-
BigStalker	50	15	3	-
MiniStalker	20	8	7	-
Shooter	100	-	-	-
Drone	60	-	-	-
Ghost	30	5	6	-
Finder	50	3	10	-
Spawnling1	500	20	5	5
MiniHunter	20	6	6	-
Spawnling2	500	20	5	4
Drone (Hunter)	30	-	-	-

Reinforcement Learning and ML-Agent

Reinforcement Learning (RL) ist ein Teilgebiet des maschinellen Lernens, bei dem ein Agent durch Interaktion mit einer Umgebung lernt, optimale Handlungen auszuführen.

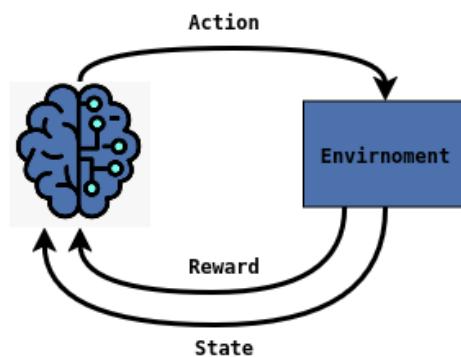


Diagramm von Reinforcement Learning

Das gezeigte Diagramm stellt den grundlegenden Kreislauf des Reinforcement Learning (RL) anschaulich dar:

- **Links:** Das Gehirn-Symbol repräsentiert den „Agenten“. Dieser Agent trifft Entscheidungen auf Basis seiner Beobachtungen und Strategie (Policy).
- **Rechts:** Das Rechteck steht für die „Umgebung“ (Environment), mit der der Agent interagiert.

Der Prozess läuft in folgenden Schritten ab:

1. **Agent:** Das lernende „Subjekt“, das Aktionen ausführt.
2. **Action (Aktion):** Der Agent führt eine Aktion aus, um die Umgebung zu beeinflussen.
3. **Environment (Umgebung):** Die Umgebung verändert sich durch die Aktion und sendet daraufhin Informationen zurück.
4. **Reward (Belohnung):** Der Agent erhält ein Belohnungssignal, das anzeigt, wie gut oder schlecht seine Aktion im aktuellen Kontext war.
5. **State (Zustand):** Zusätzlich erfährt der Agent den neuen Zustand der Umgebung.
6. **Ziel:** Die kumulative Belohnung über Zeit maximieren.

MiniGames und Rätsel-Mechaniken

Das Projekt enthält zwei Minispiele, die den Spielern zusätzliche Chancen und Belohnungen bieten. Mit dem Digit Drawer können Spieler Zahlen auf eine Canvas zeichnen, die dann von der KI erkannt werden, um Fragen im Spiel zu beantworten. Der Glyph Drawer erweitert dieses Prinzip auf komplexere Symbole (Glyphen), die die Spieler im Dungeon entdecken und dann korrekt nachzeichnen müssen. Durch das Erfüllen dieser Minispiele erhalten die Spieler zwei besondere Items, die im Spiel verwendet werden können und so ihren Fortschritt erheblich beeinflussen.

Prinzip des Zeichnens auf dem Canvas

Im MiniGame gibt es ein Canvas, auf dem der Spieler mit der Maus malen kann. Das Prinzip ist wie folgt:

- **Canvas-Initialisierung:**
 - Zu Beginn wird eine 2D-Textur mit weißer Füllung erstellt, die die Pixel des Canvas repräsentiert.
- **Zeichnen:**
 - Wenn der Spieler die linke Maustaste gedrückt hält, wird die Position der Maus auf die Canvas-Oberfläche transferiert.
 - Von der Mausposition aus wird ein Strahl in die 3D-Spielwelt geschossen (ScreenPointToRay).
 - Physics.Raycast berechnet den Treffpunkt auf der Canvas.
 - Der Weltpunkt wird in lokale Canvas-Koordinaten umgewandelt.
 - Die lokalen Koordinaten werden auf die Pixel der Textur übertragen.
 - Die Pixel innerhalb des Radius der Pinselgröße werden schwarz gefärbt.
 - Anschließend werden die aktuelle für glatte Linien und die letzte Pixelposition interpoliert.

- **Textur-Update:**
 - Nach jeder Änderung wird die Textur aktualisiert und auf das Canvas-Material angewendet und der Spieler sieht die gezeichneten Linien **in Echtzeit**.
- **Vorverarbeitung für die KI:**
 - Vor der Klassifizierung wird das gezeichnete Bild auf 28 x 28 Pixel skaliert.
 - Bei Ziffern werden die Farben invertiert, damit die KI die Ziffer richtig erkennt.

Digit Drawer

Hier werden mehrere Fragen gestellt. Jede Frage erfordert eine numerische Antwort (0–9). Der Spieler zeichnet die Zahl auf dem Bildschirm. Die KI erkennt die gezeichnete Zahl und die Anzeigetafel zeigt die Entropie jeder Ziffer im Detail an. Wenn der Spieler alle Fragen richtig beantwortet, erhält er den sogenannten "**The One Ring**", der den Spieler in Bezug auf Schaden und Geschwindigkeit stärker macht.

Glyph Drawer

Jede Aufgabe verlangt das Zeichnen von Glyphen, die bestimmten Symbolen oder Elementen in einer Liste zugeordnet sind und die auf einem Board dargestellt werden.

Glyph name	Air	Fire	Water	Earth	Energy	Time	Power	Light
Glyph sign	.((•	⊕	~	▽▽	.○	X	⤒	⤓

Um zu wissen, wie eine Glyphe aussieht, muss der Spieler die entsprechenden Items im Dungeon finden. Erst durch das Finden dieser Items kann der Spieler die Glyphen korrekt zeichnen. Sie werden nach der Aufsammlung in Inventory verfügbar sein. Der Spieler zeichnet die Glyphen auf dem Canvas. Die KI erkennt die gezeichnete Glyphe. Bei korrekter Zeichnung erhält der Spieler eine Belohnung (z.B. Entfernen der Glyphe aus der Aufgabenliste, Fortschritt im Spiel, Spawn eines Objekts).

Stats

Zuerst hatten wir ein Health-System, welches sich nur mit der Gesundheit beschäftigt hat und kein zentrales Interface bot. Da ein zentrales Interface für die Gesundheit von Spieler und Gegnern aber von Vorteil ist, haben wir uns entschieden, ein neues System zu entwickeln. In der Entwicklung des neuen Systems kam die Überlegung auch, eine zentrale Stelle für den Schaden und die Geschwindigkeit zu machen. Deshalb wurde das Stat-System entwickelt.

Das Stat-System ist eine Klasse, die zwei Listen von Floats verwaltet. Die eine Liste ist für die maximalen Stats, die andere ist für die momentanen Stats. Zur Verwaltung dieser Listen sind Getter- und Setter-Methoden implementiert und eine Methode, die beim Ausrüsten von Items aufgerufen wird.

Dabei war wichtig darauf zu achten, dass eine Erhöhung der momentanen Stats nicht den Wert des Maximums überschreitet, um Fehlverhalten vorzubeugen. Außerdem wurde darauf

geachtet, dass Stats nicht negativ werden können, weil dadurch Probleme entstehen könnten (Beispielsweise würden die Gegner stärker, je mehr sie abgeschossen werden).

Die Methode für das Ausrüsten von Items behält im Gegensatz zu den einfachen Setter-Methoden den prozentualen Wert des Stats bei. Dafür haben wir uns entschieden, dass es nicht möglich ist, sich dadurch zu heilen, dass man eine Rüstung immer wieder an- und auszieht. Aber auch, damit nicht durch das Abrüsten der Tod des Spielers verursacht werden kann.

Items

Die Items sind Gegenstände, die im Dungeon verteilt werden und dem Spieler bestimmte Vorteile verschaffen können oder essentiell für den Fortschritt sind.

Allgemeines

Alle Item-Typen haben ihre eigene Unterklasse der abstrakten Item-Klasse. Die Item-Klasse ist eine Scriptable-Object-Klasse, das bedeutet, dass Objekte dieser Klasse von überall in dem Spiel erreichbar sind und deshalb als Datenklassen sehr gut geeignet sind. Alle Items haben gemeinsam, dass sie einen Namen, eine Seltenheit, ein Spielobjekt, ein Icon haben und eine „Use“-Methode implementieren müssen.

Der Name spiegelt wider, wie das Item angezeigt wird, aber er wird auch für den Vergleich von Items verwendet. Er besitzt damit die Funktion einer ID.

Die Seltenheit sorgt dafür, wie wahrscheinlich ein Item im Dungeon erscheint und wird durch eine Farbe gezeigt:

- Grün := Gewöhnlich
- Blau := Ungewöhnlich
- Lila := Selten
- Gelb := Sehr selten

Das Spielobjekt ist eine Referenz auf ein „Prefab“, welches in der Spielwelt erzeugt und aufgesammelt werden kann.

Das Icon ist in der Ansicht des Inventars zu sehen und macht es dem Spieler leichter, die Gegenstände sofort zu erkennen.

Die „Use“-Methode muss von jeder Unterklasse selbst mit Funktionen versehen werden.

Equipment

Die erste Unterklasse ist das Equipment. Das Equipment erhält, neben den von der Item-Klasse geerbten Werten, noch eine Liste von Floats, die die Erhöhung der Stats angibt und einen Integer-Wert zwischen 0 und 5, dieser gibt an, wo die Ausrüstung angelegt wird. Die „Use“-Methode dieser Klasse ist für das Ausrüsten und Ablegen der Gegenstände verantwortlich und die damit einhergehende Erhöhung oder Reduzierung der Stats.

Beim Equipment ist die Spitzhacke als besonderes Item noch hervorzuheben, weil sie es ermöglicht, bestimmte Wände innerhalb des Dungeons zu zerstören, um so neue Wege und Räume zu erreichen. Teilweise ist es so, dass die Spitzhacke essentiell zum Fortschritt ist, weil der Raum mit dem Boss-Schlüssel ohne Tür generiert werden kann.

Boss-Schlüssel

Die zweite Unterkategorie ist der Boss-Schlüssel. Dieser erhält keine neuen Werte, aber er implementiert in seiner „Use“-Methode die Logik, um den Bossraum zu öffnen und dabei verbraucht zu werden.

Consumables

Die dritte Unterkategorie sind die Consumables. Sie repräsentieren alle Items, die der Spieler benutzt, um einen oder mehrere Stats aufzufüllen. Dazu erhalten die Consumables einen Integer, der bestimmt, welcher Stat erhöht wird, und einen Float, der über die Menge entscheidet, die dem Stat hinzugefügt wird. Ähnlich wie beim Boss-Schlüssel wird durch die „Use“-Methode hier das Item verbraucht, aber anstatt eines Raumes zu öffnen, ist hier die Logik zum Erhöhen der Stats zu finden.

Glyphs

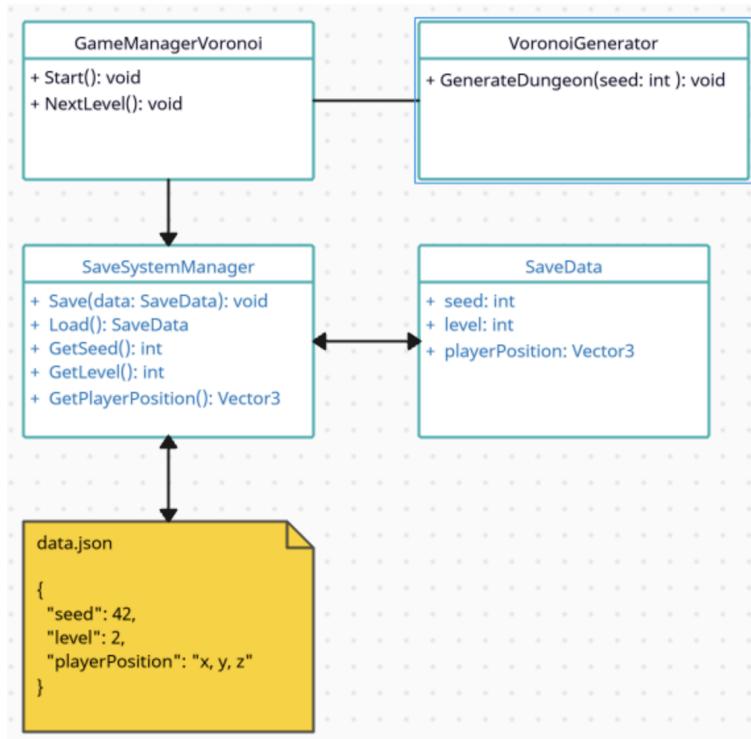
Die vierte und letzte Unterkategorie ist die Glyphs-Klasse. Diese Klasse implementiert keine Funktion für die „Use“-Methode und erhält keine neuen Werte. Sie dient lediglich als Daten-Klasse für die Glyphen des Minispiele. So ist es möglich, dass der Spieler die Glyphen in der Spielwelt finden und beim Minispiel zeichnen und damit verlieren kann.

Saving-System

Das Saving-System basiert auf einer zentralen Datenstruktur namens SaveData, die den gesamten Spielfortschritt speichert und vom SaveSystemManager bereitgestellt wird. In SaveData liegen der Seed für die prozedurale Erzeugung und die aktuelle Ebene, außerdem der Dungeonzustand mit aktueller Raumkennung, Besuchsflags für alle Räume, dem Öffnungszustand der Boss-Tür und einem Flag für den abgeschlossenen Bosskampf. Zerstörbare Wände erhalten fortlaufende Indizes, zu jedem Eintrag sichern wir Aktivität und Trefferpunkte. Für die Minispiele halten wir je ein Flag, ob die Aufgabe bereits gelöst wurde. Zum Spielerzustand speichern wir Position, Blickrichtung des Spielers und Blickrichtung der Kamera sowie zwei Listen für aktuelle und maximale Charakterwerte. Sammelbare Welt-Items verwalten wir über eine boolesche Liste, damit eingesammelte Objekte nach dem Laden nicht erneut erscheinen. Inventar und Ausrüstung liegen als feste Felder mit ItemInstance vor und werden vom SaveSystemManager bei Bedarf in ein vier-mal-fünf Raster für das Inventar und ein drei-mal-zwei Raster für die Ausrüstung übersetzt und beim Speichern wieder in die Arrays zurückgeführt.

Die Ablage erfolgt als JSON in einer Datei namens save.json im persistenten Datenordner der Engine. Der SaveSystemManager lädt diese Datei beim Start, deserialisiert sie mit JsonUtility und stellt das SaveData-Objekt global zur Verfügung. Existiert noch kein Spielstand, wird ein neuer Datensatz angelegt. Der GameManagerVoronoi liest zuerst den

Seed, erzeugt damit den Dungeon und überträgt die gespeicherten Besuchsmarken auf die Räume. Passt die Länge der Besuchsliste nicht zur neuen Dungeon-Größe, wird die Liste neu initialisiert und der Raum als besucht gesetzt. Beim Raumwechsel schreibt der Manager die neue Raumkennung und die Besuchsmarke zurück. Wird die Boss-Tür mit dem Schlüssel geöffnet, setzt der Manager das entsprechende Flag und nach dem Sieg das Flag für den abgeschlossenen Bosskampf, sodass beim nächsten Laden die Boss-Türen sofort wieder öffnen und der Level-Exit erscheint. Zerstörbare Wände und Items lesen beim Erzeugen ihren Index, prüfen ihren gespeicherten Zustand und aktualisieren beim Treffer oder beim Einsammeln die Einträge. Für einen neuen Durchlauf erzeugt StartNewRun einen frischen Spielstand mit Startwerten. Beim Wechsel auf die nächste Ebene erhöht AdvanceLevel die Ebenenzahl, setzt einen neuen Seed und leert alle pro Ebene relevanten Felder wie besuchte Räume, Zustände der zerstörbaren Wände, Item-Aktivität und Minispiel-Flags, während Inventar und Ausrüstung erhalten bleiben. Zugriffe auf Listen sind gegen ungültige Indizes abgesichert und erweitern die Strukturen bei Bedarf, damit neue Einträge sicher gespeichert werden.



Alternativen und Entscheidungen

Spielengine

Eine Spielengine ist eine Software und zugleich Framework, welche den Entwicklern einige Programmier- & Handhabungsaufgaben abnimmt. Sie bietet in der Regel eine Benutzeroberfläche, um Objekte oder Skripte direkt an andere Objekte oder Skripte anzuhängen oder zu übergeben. In manchen Bereichen dieser Engines wird *visual scripting* ermöglicht, was meist keine direkten Programmierkenntnisse benötigt, sondern durch Bausteine und die Verknüpfung dieser werden Programme oder Skripte erstellt. Zur Auswahl an Spielengines standen für uns die Unity und Unreal Engine.

Unity

Die Spielengine, die von Unity Technologies entwickelt wird, heißt Unity. Diese Engine verwendet die Programmiersprache C# und bietet auch begrenztes visual scripting an, meist aber für Shader. Sie ist kostenlos verfügbar, solange man sie im nicht-kommerziellen Bereich verwendet oder als Schüler und Student. Ansonsten müssen Lizenzgebühren bezahlt werden, dieses System ist umfangreicher aufgestellt, ist aber für unseren Fall nicht von Bedeutung. Gleichzeitig kommt die kostenpflichtige Variante auch mit weiteren Funktionsumfängen, die für unseren Anwendungsfall aber nicht notwendig sind. Anwendung findet sie in vielen Spielen, meist von unabhängigen Entwicklern. Einige Gruppenmitglieder haben bereits vor Projektstart Erfahrung mit dieser Engine sammeln dürfen.

Unreal Engine

Epic Games Spielengine nennt sich Unreal Engine. Diese Firma entwickelte auch Spiele, wie Fortnite oder Rocket League. Diese Engine findet auch dort direkt Anwendung, wodurch diese von der entwickelnden Firma direkt auf die Anwendung erprobt wird. Die Programmiersprache hier ist C++, allerdings ist hier auch die Möglichkeit komplett darauf zu verzichten und alles mit Hilfe von visual scripting. Die Community hier ist kleiner als die von Unity, vermutlich aufgrund der Programmiersprache C++. Hier werden Lizenzgebühren erst ab einem Umsatz von 1 Millionen US-Dollar fällig in Höhe von 5%. Mit dieser Engine hatte keiner bisher gearbeitet.

Entscheidung

Wir haben uns aufgrund der Vorerfahrung von einigen Gruppenmitgliedern für die Unity Engine entschieden, da dies die Einarbeitungszeit abnimmt und gleichzeitig den unerfahrenen Gruppenmitgliedern Unterstützung geboten werden kann.

Projektvorgehensmodell

Zu Beginn haben wir uns kurz über die Möglichkeit zwischen agilen und iterativen Vorgehen beraten und sehr schnell auf den gemeinsamen Nenner gekommen, dass wir agil vorgehen wollen, da diese Varianten in der Industrie verwendet werden.

Scrum

Bei Scrum gibt es Sprints, die zeitlich terminiert sind, in der Regel dauern diese 2 bis 4 Wochen an. Aufgaben werden über einen Backlog gepflegt, welcher einen Aufgabenkatalog darstellt, die es zu erledigen gilt bis zum Ende des Projekts oder eines Zyklus des Produkts. und aus welchem die Aufgaben für den Sprint definiert werden. Täglich finden auch 15 minütige Meetings statt, die zur Besprechung von möglichen Schwierigkeiten oder Problemen der Aufgaben dienen. Man versucht hier die Planung von Aufgaben maximal in der Größe eines Sprints zu halten, damit diese schaffbar sind. Sind diese zu groß, werden sie weiter unterteilt. Diese Variante eignet sich sehr gut bei großen, komplexen und langfristigen Projekten, auch für die kontinuierliche Produktentwicklung, solange das Ziel ist den Prozess effizienter zu gestalten.

Kanban

Aufgaben werden bei Kanban ebenfalls mit einem Backlog gepflegt, allerdings gibt es hier keine festgelegten Sprints, sondern es werden Aufgaben wie bei einer To-Do-Liste abgearbeitet. Dabei werden sie zu Beginn aus dem Backlog herausgenommen, sobald man diese anfängt zu bearbeiten und verschiebt diese je nach Organisation weiter, je weiter der Fortschritt der Aufgabe voranschreitet. Standard oder Minimum sind dabei Bearbeitung und Erledigt als Zustände, die durch Spalten am Kanbanboard, dem "Schwarzen Brett" der Aufgaben, dargestellt werden. Die erste Spalte ist dabei der Backlog, aus dem immer die Aufgaben herausgenommen werden. Dieses Vorgehen eignet sich gut für Support- und Wartungsteams, aber auch für die Produktentwicklung.

Scrumban

Scrumban setzt sich aus den Worten Scrum und Kanban zusammen und will die Vorteile aus beiden Modellen vereinen: die Flexibilität durch das Board von Kanban und die Grundfunktionen von Scrum. Es gibt auch ein Schwarzes Brett wie bei Kanban, das ebenfalls individuell erweitert werden kann, also gibt es keine Unterschiede dahingehend. Verwendet wird Scrumban gerne von Startups verwendet oder für die Produktentwicklung.

Entscheidung

Am Ende des Tages kann man sich für jedes Vorgehensmodell entscheiden, da dort der Teufel im Detail liegt. Jedes Team hat seine Eigenheiten im Vorgehen, weil jedes Team andere Bedürfnisse und Menschen hat, somit gibt es exakte Varianten, wie Sand am Meer. Wir haben uns allerdings für Scrumban entschieden, da es eben für Startups und somit auch für kleinere Teams gut geeignet ist. Gleichzeitig wollten wir auch keine täglichen Meetings haben, da das unseren Rahmen sprengen würde und man vermutlich nicht täglich am Projekt arbeiten würde, im Gegensatz zu einer Vollzeitstelle in der Wirtschaft.

Versionsverwaltungssystem

Versionierungssysteme sollen die Entwickler unterstützen, kollaborativ an einem Projekt arbeiten zu können, um somit die Effizienz zu steigern. Bei den Varianten gibt es verschiedene Herangehensweisen. Beide folgenden Optionen sind freie Software und uneingeschränkt nutzbar.

Git

Git ist ein verteiltes Versionierungsverwaltungssystem, das am meisten in der Wirtschaft verwendet wird. Es erlaubt, eine Kopie des gesamten Projekts auf dem eigenen Gerät zu speichern, ohne Einschränkungen hinsichtlich der Schreiberlaubnis zu haben. Es verfolgt Änderungen anhand der Inhalte der Dateien. Man kann es über die Kommandozeile bedienen oder über eine Benutzeroberfläche, dafür gibt es verschiedene Möglichkeiten und explizite Software, welche die Bedienung erleichtern soll. Einige Gruppenmitglieder haben bereits vor Projektstart Erfahrung mit Git sammeln dürfen.

SVN

SVN ist ein zentrales Versionierungsverwaltungssystem. Alle Benutzer können auf das Projekt zugreifen. Hier werden Änderungen über die Dateien selbst aufgenommen, das bedeutet, dass nicht allein der Inhalt, sondern die Änderung von Dateien anhand des Pfades gespeichert werden. Daraus resultiert, dass der erste Benutzer, der eine Datei bearbeitet, diese blockiert, bis seine Änderungen veröffentlicht wurden im zentralen Projekt. Erst dann kann jemand anderes die Datei verändern. Auch hierfür gibt es Benutzeroberflächen, die einem die Bedienung angenehmer gestalten.

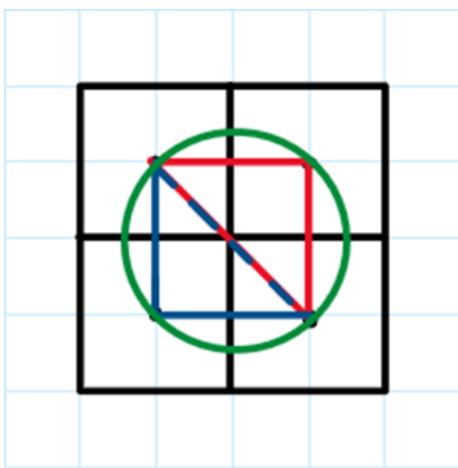
Entscheidung

Ähnlich zu der Wahl bei der Spielengine, haben wir bei dem Versionierungsverwaltungssystem für die bereits bekannte Variante entschieden, da noch nicht alle Mitglieder Erfahrungen mit dem kollaborativen Arbeiten sammeln durften über ein solches System. Ein weiterer Aspekt war die aktuelle Anwendung in der Industrie, deshalb fiel die Entscheidung hier auch sehr schnell und leicht auf Git.

Probleme & Lösungen

Unsaubere Delaunay-Triangulierung

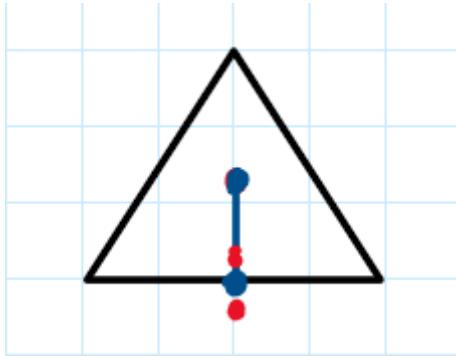
Bei einer Delaunay-Triangulierung wurde im Kapitel Dungeon-Generierung geschrieben, dass ein Delaunay Dreieck als verletzt gilt, wenn ein Punkt, der nicht zum Dreieck gehört, in dessen Umkreis liegt. Es kann aber von Nöten sein, dass diese Definition so erweitert wird, dass ein Punkt, der genau auf dem Umkreis liegt, nicht als innerhalb gilt und deshalb eine valide Delaunay-Triangulierung liefert. Siehe im Beispiel:



In diesem Bild sieht man in Schwarz ein Quadrat zur Begrenzung und die Linien, die das Voronoi-Diagramm liefern würde. In Rot ist ein Dreieck der Triangulierung zu sehen, in Blau ein zweites. In Grün wird der Umkreis der beiden Dreiecke gezeigt, da beide denselben Umkreismittelpunkt besitzen. Dadurch liegt immer ein vierter Punkt auf dem Umkreis, was eigentlich nicht erlaubt ist, aber oftmals hingenommen wird. Ohne diese Heuristik wäre es sehr komplex gewesen, bei zufällig platzierten Punkten zu garantieren, dass eine Triangulierung überhaupt möglich ist.

Verlängern der äußeren Voronoi-Kanten

Das Verlängern der Kanten im Voronoi-Diagramm Richtung Unendlich erwies sich als sehr schwieriges Problem. Es kam zum Problem, wenn der Umkreismittelpunkt außerhalb des Dreiecks lag, weil dann die Kante in die falsche Richtung verlängert wurde. Einige der ersten Versuche haben einige Probleme gelöst aber nach kurzer Zeit und bestimmten Punkteverteilungen traten Fehlgebildete Kanten auf. Nach viel Zeit und Gedanken ist dann aber die Lösung gefunden worden. Der Kerngedanke wird in dieser Zeichnung deutlich:



Wir sehen das Dreieck in Schwarz und die Mittelsenkrechte in Blau. Die beiden roten Punkte liegen auf der blauen Geraden, einer der Punkte liegt im Dreieck, der Andere liegt außerhalb des Dreiecks. Wir nehmen dann die Richtung vom Mittelpunkt der Dreiecksseite zu dem Punkt, der außerhalb des Dreiecks ist, gehen dann aber vom Umkreismittelpunkt aus in diese Richtung. Zunächst wird die Kante sehr lang gemacht, um auf jeden Fall die Dungeon-Grenze zu berühren. Danach wird die Kante auf den Schnittpunkt mit der Dungeon-Grenze gekürzt.

Voronoi-Kanten falsch generiert

Einige Kanten des Voronoi-Diagramms sind in die falsche Richtung generiert worden. Auch wenn der Fall sehr selten aufgetreten ist, haben wir versucht, das Problem zu finden und zu lösen. Das Problem war relativ schnell gefunden und gelöst. Das Problem war, dass bei den Vergleichen in der Point- und Edge-Klasse die genaue Äquivalenz überprüft wurde. Das ist bei Floats ein Problem, da immer Rundungsfehler auftreten können. Deshalb existiert in C# eine Methode, die zwei Floats vergleicht auf Basis einer kleinen Grenze, ab wann zwei Zahlen ungefähr gleich sind.

Spieler Kamera laden

Bei der Speicherung der Rotation der Spieler Kamera entstand das Problem, dass die Kamera teilweise nicht richtig rotiert wurde. Das Problem tauchte aber nicht auf jedem Gerät gleichmäßig auf, sondern am schlimmsten auf Windows-Rechnern und sehr wenig auf Linux- und Mac-Rechnern. Das Problem bestand in den Ladezeiten der Szene. Die Windows Rechner benötigen generell mehr Zeit. Dadurch wurde das Kamera-Skript bereits geladen und es war möglich, die Kamera schon zu bewegen, ohne dass das Spiel geladen wurde. Gelöst wurde das Problem, indem das Kamera-Skript erst nach Abschluss des Ladevorgangs aktiviert wird.

Gebrochene Referenzen

Gegen Ende haben wir die Ordnerstruktur aufgeräumt und dabei Ordner und Skripte umbenannt sowie Assets in neue Bereiche verschoben. Dadurch änderten sich an mehreren Stellen GUIDs und Klassennamen und Unity konnte Referenzen in Szenen, Prefabs und ScriptableObjects nicht mehr auflösen. In der Folge erschienen fehlende Komponenten und leere Felder, Türen und Spawner verloren ihre Verknüpfungen und einzelne Manager ließen sich nicht mehr instanziieren. Wir haben alle betroffenen Szenen und Prefabs geöffnet, fehlende Skripte ersetzt, Referenzen zu Prefabs, ScriptableObjects und Events neu gesetzt und anschließend einen klaren Ablauf festgelegt. Verschieben und Umbenennen erfolgt nur noch im Unity Editor, Meta Dateien bleiben im Versionskontrollsystem erhalten, Klassen werden immer zusammen mit dem Dateinamen umbenannt und größere Änderungen werden in kleinen Schritten mit Zwischenläufen vorgenommen. So lassen sich gelöste Referenzen zuverlässig wiederherstellen und erneute Ausfälle vermeiden.

Nutzung & Steuerung

Spielstart

Nach dem Programmstart erscheint der Startbildschirm mit den drei Schaltflächen Start, Laden und Beenden.

Start beginnt eine neue Runde. Dabei wird ein zufälliger Seed erzeugt. Dieser Seed steuert die komplette prozedurale Erstellung des Levels. Er legt die Grundfläche, die Verteilung der Räume und ihre Verbindungen fest. Er bestimmt außerdem, welche Items erscheinen, welche Glyphen im Rätsel verwendet werden und welche Zahlen Fragen im Digit-Minispiel gestellt werden. Zusätzlich wählt er eine Skybox und eine Bodenvariante aus und streut Umgebungselemente wie Gras, Büsche, Bäume und Felsen im Spielbereich. Auch die Zusammensetzung der Gegner in Gegner-Räumen sowie die Auswahl des Bosses werden vom Seed beeinflusst. Nach der Generierung wechselt das Spiel in die Dungeon-Szene. Der Spieler spawnt im Startraum. Die Spielerwerte Gesundheit, Schaden und Bewegungsgeschwindigkeit starten in ihrem Standardzustand für eine neue Runde.

Laden lädt einen vorhandenen Spielstand und wechselt in die Dungeon-Szene. Dabei werden Ebene und Seed, die besuchten Räume inklusive aktuellem Aufenthaltsraum, Türzustände inklusive Boss-Tür, der Zustand der zerstörbaren Wände, die Position und Blickrichtung des Spielers, die Kameraausrichtung sowie Inventar, Ausrüstung und die aktuellen Spielerwerte wiederhergestellt. Ist kein gültiger Speicherstand vorhanden, bleibt man im Startmenü.

Beenden beendet das Spiel direkt aus dem Startbildschirm.



Steuerung

Aktion	Taste
Bewegen	W A S D
Umschauen	Maus bewegen
Schießen	Leertaste
Inventar öffnen oder schließen	I
Im Inventar ausgewähltes Item benutzen oder ausrüsten	E
Im Inventar ausgewähltes Item entfernen	O
Karte ein oder aus	M
Pause-Menü öffnen oder schließen	P
Minispielansicht betreten am Zeichenboard	G
Minispielansicht verlassen zurück zur First Person Ansicht	Tab
Auf dem Zeichenboard zeichnen	Linke Maustaste halten
Zeichen auf dem Zeichenboard auswerten	Rechte Maustaste
Zeichenfläche leeren	C
Spitzhacken Schlag an brüchiger Wand ausführen	Rechte Maustaste

Inventar

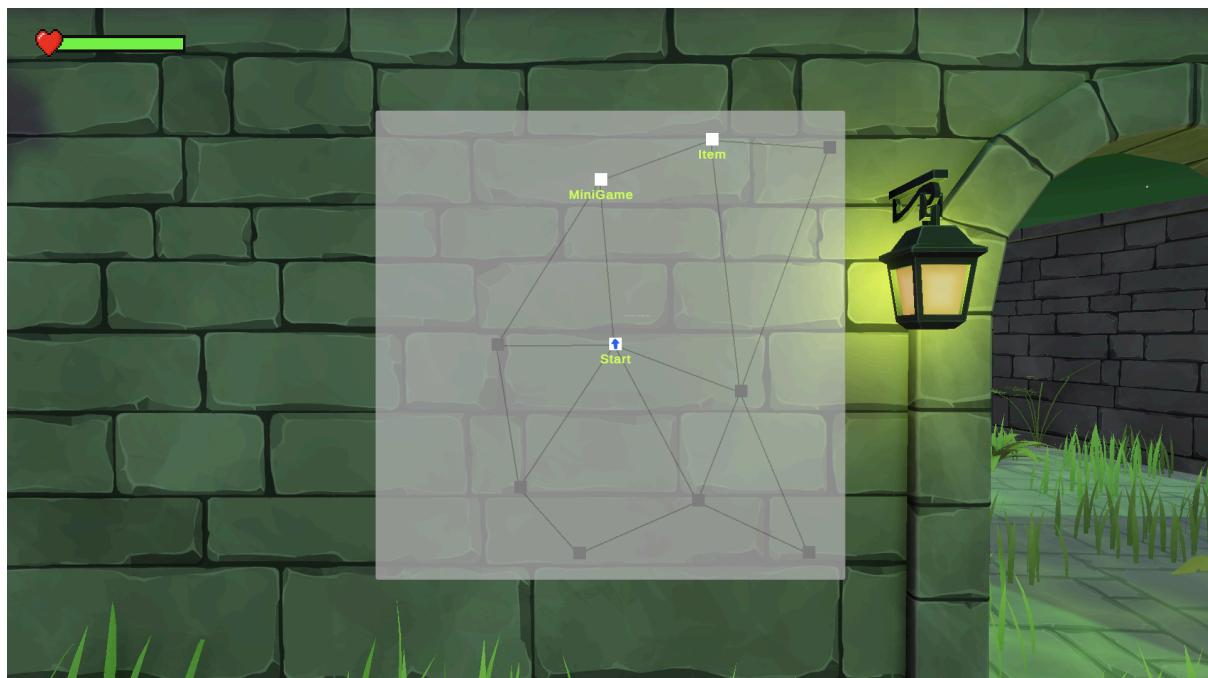
Das Inventar lässt sich mit der Taste "I" öffnen und schließen, dabei wird die Zeit im Spiel angehalten, sodass keine Interaktion in der Spielwelt möglich ist, sondern nur im Inventar. Der Mauszeiger wird sichtbar und kann frei bewegt werden. Beim Schließen wird der Mauszeiger wieder gesperrt und das Spiel läuft weiter. Bei geöffneten Inventar zeigt sich folgendes Bild



In dieser Ansicht ist es möglich, durch Linksklick einen der Slots zu markieren und dann mit "E" ein Item zu benutzen oder mit "O" ein Item aus dem Inventar zu entfernen, falls ein Item vorhanden ist. Das Verwenden der Items wird so durchgeführt, dass das Inventar die "Use"-Methode des ausgewählten Items aufruft. Das Entfernen der Items löscht ein Item aus dem Item Slot, sollte der Slot danach leer sein, wird er zurück auf "NULL" gesetzt.

Karte

Die Karte wird mit M ein- und ausgeschaltet. Sie zeigt den Dungeon als Graph aus kleinen Quadraten, nicht als Innenraumansicht. Jedes Quadrat steht für einen Raum. Besuchte Räume sind klar dargestellt und tragen eine Beschriftung mit ihrem Typ, zum Beispiel Start, Normal, Item, MiniGame oder Boss. Unbesuchte Räume sind ausgegraut. Linien zwischen Quadraten zeigen die möglichen Verbindungen zu Nachbarräumen. Das aktuell belegte Quadrat enthält einen Pfeil, der die Blickrichtung des Spielers anzeigt. Das Spiel läuft während der Kartenansicht weiter. Im Pause-Menü und bei Game Over ist die Karte nicht verfügbar.



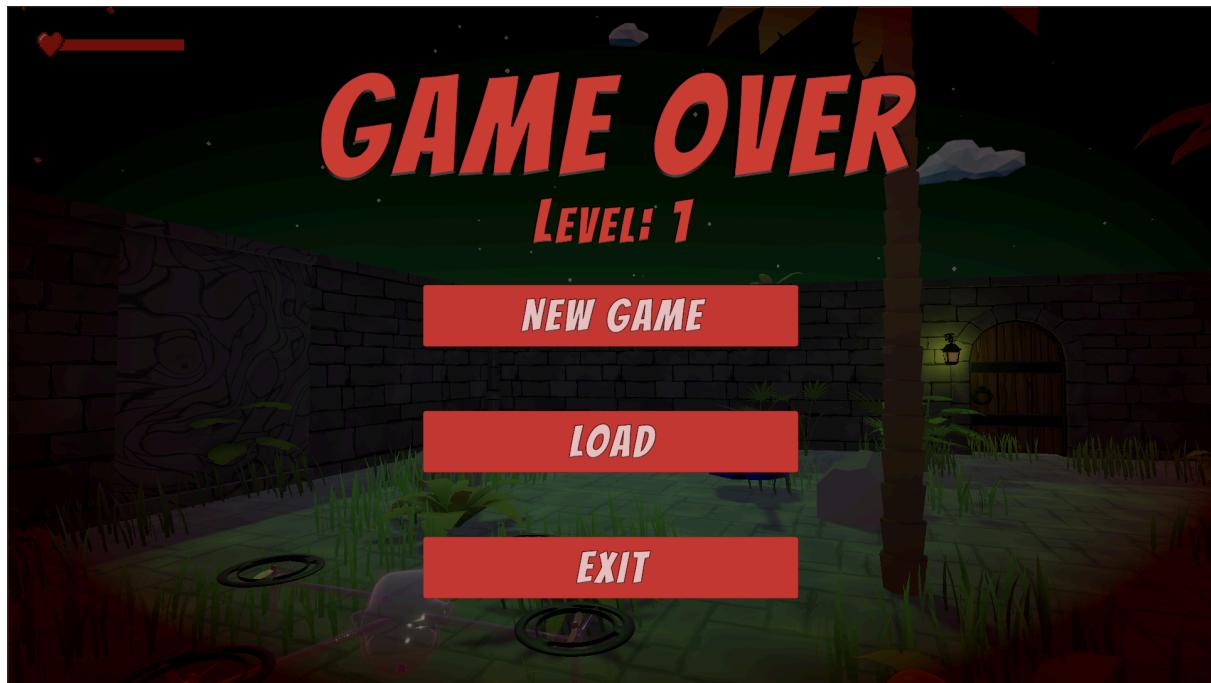
Pause-Menü

Das Pause-Menü wird mit P geöffnet und geschlossen. Beim Öffnen pausiert das Spiel und der Mauszeiger wird sichtbar. Aus dem Pause-Menü heraus kann gespeichert, ein vorhandener Spielstand geladen oder das Spiel beendet werden. Während aktiver Kämpfe ist das Pause-Menü gesperrt und wird automatisch geschlossen, bis der Kampf vorbei ist. Die Karte ist im Pause Menü oder im Game Over Menü nicht verfügbar. Beim Verlassen des Menüs wird der Mauszeiger wieder gesperrt und das Spiel fortgesetzt.



Game Over Menü

Die Game Over Ansicht erscheint, wenn die Gesundheit auf null fällt. Er zeigt die erreichte Ebene und bietet drei Optionen: Neues Spiel, Laden und Beenden. Neues Spiel startet eine frische Runde mit neuem Seed und Standardwerten. Laden stellt den letzten Spielstand her. Beenden schließt die Anwendung. In der Game Over Ansicht ist die Figur gesperrt. Die Zeit steht still. Der Mauszeiger ist sichtbar. Karte, Inventar und Pause-Menü sind deaktiviert.



Minispiel Ansicht

In der Nähe eines Zeichenbereichs für Ziffern oder Glyphen wechselst du mit G in die Zeichenansicht. Dabei wird eine separate Kameraansicht aktiviert, die sich ausschließlich auf die Zeichenfläche konzentriert und kurze Anleitungen einblendet. Du zeichnest mit gedrückter linker Maustaste, mit C leerst du die Fläche. Mit der rechten Maustaste lässt du die aktuelle Zeichnung auswerten. Das System zeigt die Vorhersage an und vergleicht sie mit der erwarteten Eingabe. Mit Tab kehrst du jederzeit in die First-Person-Ansicht zurück. Während der Zeichenansicht ist die Spielerbewegung deaktiviert, damit der Fokus vollständig auf dem Zeichnen liegt. Beim Betreten erscheinen die Hinweise automatisch, beim Verlassen des Interaktionsbereichs blenden sie sich wieder aus, wie in den beiden Abbildungen direkt darunter zu sehen ist.



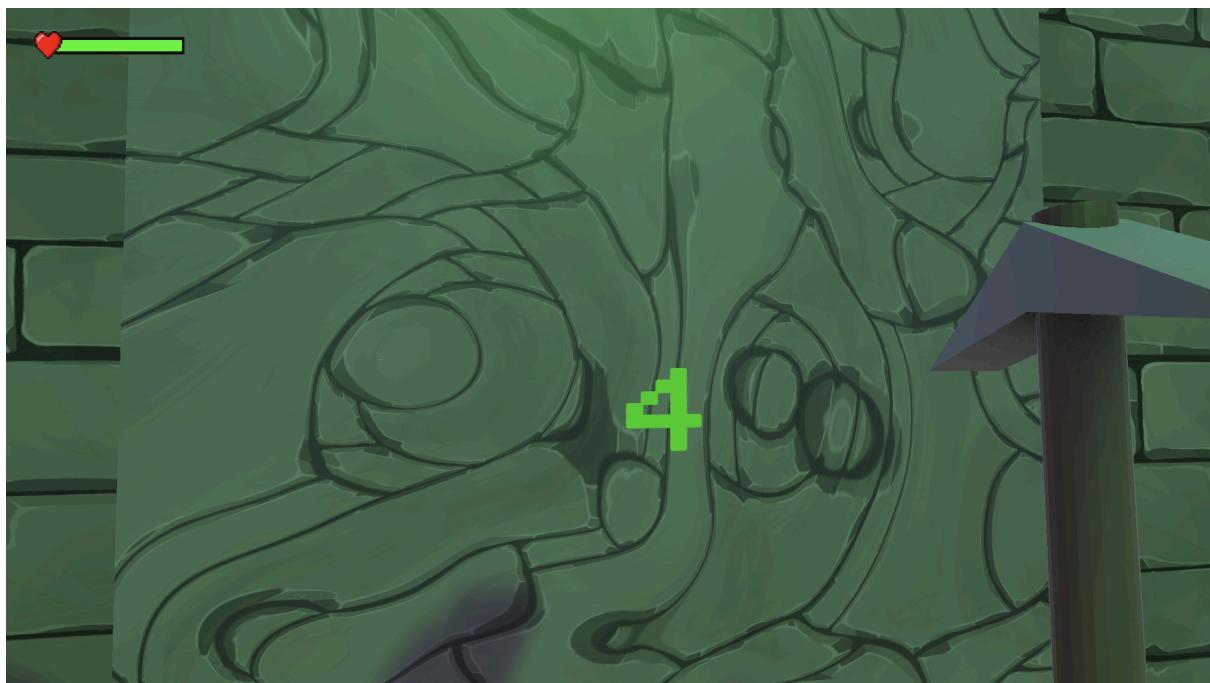
Kampf und Schießen

Geschossen wird mit der Leertaste. Das Projektil folgt der aktuellen Blickrichtung des Spielers. trifft ein Projektil auf ein Hindernis oder endet seine Flugzeit, verschwindet es automatisch. Die Projektile werden aus einem Objektpool entnommen und am Schusspunkt unmittelbar aktiviert. Dadurch bleiben die Schüsse flüssig und performant, auch bei schneller Abfolge.



Spitzhacke

An brüchigen Wänden erscheint eine Einblendung, wenn die Spitzhacke im rechten Hand Slot ausgerüstet ist. Der Schlag wird mit der rechten Maustaste ausgelöst. Mehrere Schläge reduzieren die Haltbarkeit der Wand, bis sie verschwindet und der Durchgang frei wird. Ein kurzer Hinweistext begleitet den Ablauf und blendet sich nach kurzer Zeit wieder aus.



Interaktionen

Im Spiel interagiert man auf verschiedene Arten. Dazu gehören das Annähern an Objekte, das automatische Aufheben von Items, der Einsatz von Ausrüstung oder Verbrauchsgegenständen, das Lösen der Minispiele und Kämpfe. Hinweise erscheinen nur dort, wo sie wirklich benötigt werden. An den Zeichentafeln der Minispiele blendet sich in der Nähe eine kurze Anleitung ein, ebenso bei brüchigen Wänden. Items, Türen und auch der Level-Ausgang funktionieren ohne zusätzliche Bestätigung.

Items liegen sichtbar im Dungeon. Wenn der Spieler darüber läuft, werden sie automatisch eingesammelt und ins Inventar gelegt. Verbrauchsgegenstände wie Heiltränke wirken sofort und füllen direkt die Lebenspunkte auf. Ausrüstungsgegenstände verändern Attribute wie Leben, Schaden oder Geschwindigkeit, solange sie angelegt sind. Beim Zahlen-Minispiel, dem „Digit-Drawer“, beantwortest du zehn Fragen. Das Item erhältst du erst, wenn alle Antworten richtig waren. Beim Glyphen-Minispiel, dem „Glyph-Drawer“, müssen zuerst die passenden Glyphen-Items im Dungeon gefunden werden. Diese geben vor, welche Symbole gezeichnet werden sollen. Wird eine gefundene Gyphe korrekt nachgezeichnet, verschwindet sie aus dem Inventar. Die Anzahl der geforderten Glyphen ist zufällig und wird pro Ebene festgelegt. Erst wenn alle benötigten Glyphen gelöst sind, erhältst du den Boss Schlüssel.

Die Zeichenrätsel laufen über ein Canvas. In der Nähe der Tafeln wird erklärt, wie man in die Zeichenansicht wechselt, wie man zeichnet, die Fläche leert und wieder in die First Person Ansicht zurückgeht. Beim Digit-Drawer werden Fragen beantwortet, indem Ziffern gezeichnet werden. Beim Glyph-Drawer werden die im Dungeon gefundenen Symbole nachgezeichnet. Richtig Eingaben werden sofort geprüft. Eine Belohnung gibt es erst, wenn alle Aufgaben des jeweiligen Minispiele vollständig abgeschlossen sind.

Normale Türen öffnen und schließen sich automatisch. Solange kein Kampf läuft, sind sie offen. Betritt man jedoch einen Raum mit Gegnern, schließen sich die Türen und bleiben so lange zu, bis alle Gegner besiegt sind. Bosstüren lassen sich nicht direkt öffnen. Hierfür wird der Boss Schlüssel benötigt, den man in einem Nachbarraum des Bossraums einsetzt. Danach öffnen sich die Türen und bleiben für die aktuelle Ebene offen.

Brüchige Wände können mit einer Spitzhacke zerstört werden. Voraussetzung ist, dass die Spitzhacke im Ausrüstungs-Slot angelegt ist. Steht man in der Nähe einer solchen Wand, erscheint eine kurze Einblendung, wie sie eingesetzt wird. Die Wand besitzt Lebenspunkte, die als Zahl direkt angezeigt werden. Jeder Schlag senkt den Wert, bis er null erreicht. Dann verschwindet die Wand. Ihr aktueller Zustand und die verbleibenden Lebenspunkte werden im Spielstand gespeichert, solange man sich auf derselben Ebene befindet.

Im Kampf richten sich Projektilen nach der Blickrichtung des Spielers. Ein Treffer zieht dem Gegner Lebenspunkte ab, die als Zahl über ihm sichtbar sind. Gegner verursachen Schaden, wenn sie den Spieler treffen oder berühren. Während eines Gefechts bleiben Menüs gesperrt und Türen geschlossen. Erst nach dem Sieg werden sie wieder freigegeben. Damit Schüsse flüssig bleiben, stammen die Projektilen aus einem Objektpool und werden nach der Benutzung dorthin zurückgeführt.

Nach einem besiegteten Boss erscheint ein Ausgang. Betritt man ihn, startet die nächste Ebene mit einem vollständig neu generierten Dungeon. Die neue Ebene verhält sich wie ein frischer Durchgang. Räume, Türen, Wände, Gegner und Items werden neu angeordnet. Bosstüren sind zu, als hätte man ein neues Spiel begonnen. Der Fortschritt der vorherigen Ebene beeinflusst das neue Layout nicht. Läßt man später einen gespeicherten Spielstand, landet man wieder an der zuletzt gesicherten Position der alten Ebene.

Der Gesundheitszustand wird durch eine rote Vignette am Bildschirmrand angezeigt. Sinkt die Gesundheit unter die Hälfte, färbt sich der Rand rot. Je weiter die Lebenspunkte sinken, desto stärker wird dieser Effekt. So erkennt der Spieler sofort, wenn Gefahr droht, ohne ständig auf Anzeigen achten zu müssen.

Menü & UI

Das Interface ist bewusst schlank gehalten, damit der Fokus auf der Spielwelt bleibt. Permanente UI-Elemente sind auf das Nötigste reduziert und treten nur dann in den Vordergrund, wenn es für die Orientierung sinnvoll ist. Informationen erscheinen in drei Ebenen: dauerhaft sichtbare Hinweise, kurze Einblendungen direkt in der Welt und klar abgegrenzte Menüs.

Dauerhaft sichtbar sind der Lebensbalken oben links und die rote Vignette, die bei sinkender Gesundheit stärker wird. Zahlen über Gegnern und die restlichen Lebenspunkte brüchiger Wände werden direkt in der Spielwelt eingeblendet. So bekommt der Spieler präzises Feedback, ohne dass große Elemente der Benutzeroberfläche die Sicht verdecken.

Die Menüs sind klar getrennt und verhalten sich konsistent. Es ist immer genau eine überlagernde Ansicht aktiv. Öffnest du eine andere, schließt sich die zuvor geöffnete automatisch. Es gibt den Startbildschirm mit Start, Laden und Beenden. Die Karte legt sich als Einblendung über das laufende Spiel und ist im Pause-Menü und im Game Over Bildschirm ausgeblendet. Das Inventar pausiert das Spiel, zeigt Raster und Ausrüstungsplätze und macht den Mauszeiger frei. Die Minispiel-Ansicht wechselt auf die Zeichenfläche, blendet kurze Hinweise ein, der Spieler bleibt stehen. Das Pause-Menü hält das Spiel an, zeigt die aktuelle Ebene und bietet Speichern, Laden und Beenden. Während eines Kampfes ist es gesperrt und schließt sich automatisch. Die Game Over Ansicht erscheint bei null Gesundheit, zeigt die erreichte Ebene und bietet Neues Spiel, Laden und Beenden.

Start- und Pause-Menü übernehmen die Verwaltung des Spielfortschritts. Es gibt genau einen Speicherstand. Beim Speichern wird der aktuelle Zustand in eine JSON-Datei geschrieben, nachdem die benötigten Werte ermittelt wurden, zum Beispiel Spieler Position und Blickrichtung, Kameraausrichtung, Inventar, Ausrüstung, Statistiken, Tür- und Raumzustände sowie die aktuelle Ebene. Speichern in Gegner- oder Bossräumen ist bewusst deaktiviert, um unfaire Situationen zu vermeiden und die Speicherdatei schlank zu halten. Laden stellt einen vorhandenen Spielstand her und lädt die Szene mit den gespeicherten Werten neu. Ist kein gültiger Spielstand vorhanden, bleibt man im Startmenü. Start erzeugt einen neuen Seed, generiert die Szene und legt anschließend einen initialen

Speicherstand an. Beenden schließt die Anwendung ohne zusätzliches Speichern. Nicht gesicherter Fortschritt geht verloren.

Ziel ist einfache Lesbarkeit und eine verlässliche Bedienung. Kurze Texte, dezente Einblendungen und Feedback direkt in der Welt halten den Blick auf das Spiel, und das einheitliche Verhalten der Menüs macht jederzeit klar, ob das Spiel läuft, pausiert oder Eingaben gerade an die Benutzeroberfläche gehen.

Bekannte Bugs & Hinweise

Beim Betreten von Gegner- oder Bossräumen können sich die Türen schließen, während der Spieler noch genau auf der Schwelle steht. In diesem Sonderfall lässt sich der Raum manchmal wieder nach außen verlassen, obwohl der Kampf bereits begonnen hat. Das passiert nur, wenn man sehr langsam durch die Tür geht und der Schließvorgang genau in dem Moment auslöst, in dem der Spieler noch teilweise im Türbereich steht. Problemumgehung: Räume zügig und vollständig betreten, bevor man stehen bleibt. Geplante Abhilfe: Beim Schließen prüfen, ob sich der Spieler noch im Kollisionsbereich der Tür befindet, und ihn kurz nach innen schieben, bevor die Tür schließt.

Außerdem lässt sich das Glyphen-Minispiel auch dann korrekt lösen, wenn die passenden Glyphen-Items vorher nicht eingesammelt wurden. Das verschafft keinen spielerischen Vorteil, kann aber dazu führen, dass später gefundene Glyphen-Items unnötig Inventarplatz belegen. Problemumgehung: Erst die zugehörigen Glyphen-Items einsammeln oder überflüssige Items im Inventar manuell verwerfen. Eine mögliche zukünftige Lösung wäre, vor der Wertung zu prüfen, ob das passende Item vorhanden ist.

Selten kann es vorkommen, dass prozedural erzeugte Türen schmäler sind als die effektive Breite des Spielers. Der Spieler nutzt zur Kollision einen Capsule-Cast anstelle eines starren Colliders. Die Kapselbreite wurde zwar angepasst, um Hängenbleiben an Kanten zu vermeiden, in sehr schmalen Türöffnungen passt man aber gelegentlich nicht hindurch. Problemumgehung: leicht seitlich versetzen und schräg ansetzen oder einen anderen Zugang wählen. Geplante Abhilfe: eine Mindestbreite für Türöffnungen bei der Generierung erzwingen oder die effektive Kapselbreite beim Durchqueren enger Passagen kurzzeitig verringern.

Vor Boss Räumen oder vor dem Level Ausgang lohnt sich ein manueller Speicherpunkt über das Pause-Menü. Nach dem Betreten des Ausgangs startet eine neue Ebene mit frischem Layout und neuen Gegnern, eine Rückkehr ist nicht vorgesehen. Für die Minispiele gilt: sauber und mittig zeichnen und im Zweifel mit C die Fläche leeren. Brüchige Wände reagieren nur, wenn die Spitzhacke im rechten Hand-Slot ausgerüstet ist. Erst dann erscheint die kurze Einblendung in der Nähe der Wand. Die Karte legt sich über das laufende Spiel und pausiert somit nicht, daher besser in sicheren Räumen nutzen. Während aktiver Kämpfe ist das Pausemenü gesperrt. Das Inventar pausiert das Spiel und sollte idealerweise nur genutzt werden, wenn der Raum bereits gesichert ist.

Projektverlauf

Sprint 1

(18.10.2024 - 13.11.2024)

Dieser Sprint markiert die Zusammenführung der Gruppe und den offiziellen Start der Projektgruppenarbeit.

Ziele:

- Grundlegende Diskussion und Abstimmung hinsichtlich verwendeter Technologien (Unity vs Unreal, Git vs SVN) durchführen
- Abstimmung welches Vorgehen im Projektmanagement
- Vereinbarkeit von regelmäßigen Meetings auf einheitliche Zeiträume
- Abgleichung von präferierten Aufgabenbereichen und bisheriger Kenntnisstände

Ergebnisse:

Zuerst haben grobe Absprachen stattgefunden, wie in welchem Bereich Präferenzen liegen zu arbeiten, bisheriger Kenntnisstand und weiteres.

Wir hatten uns zu Beginn darauf geeinigt, dass Meetings samstags von 9 bis 10 Uhr oder mittwochs zwischen 13 und 14 Uhr oder ab 21 Uhr stattfinden können.

Ebenfalls ist eine Einigung zustande gekommen hinsichtlich des Vorgehens im Projektmanagement. Durch die Industrieerfahrung einiger Gruppenmitglieder konnte hier sowohl auf Wissen aus der Praxis, als auch auf Vor- und Nachteile von Vorgehen zurückgegriffen werden. Prinzipiell standen agile, als auch iterative Verfahren, wobei schnell klar war, dass es agiles Vorgehen werden sollte, da diese in der Praxis verwendet werden. Hier fiel die Entscheidung auf Scrumban mit Sprints von zweiwöchiger Dauer inklusive eines Backlogs. Aufgaben sollten in der Regel innerhalb eines Sprints erledigt werden können und ein Meeting stellt sowohl den Beginn eines neuen Sprints, als auch das Ende des vergangenen Sprints dar.

Im Bereich der zu verwendenden Technologien haben wir uns für die Unity Engine und bei der Auswahl der Versionierung konnten wir uns für Git mit der zugehörigen Plattform GitHub einigen.

Sprint 2

(14.11.2024 - 29.11.2024)

Ziele:

- Einarbeitung in Unity mit Hilfe von Youtube Tutorials und oder kleiner eigener Ideen um Vertrauen in die Engine, als auch die Verwendung dieser zu schaffen

Ergebnisse

Dieser Sprint war für die Einarbeitung in Unity gedacht, um sich mit der Benutzeroberfläche, einiger grundlegender Konzepte und kleiner Tricks und Kniffe vertraut zu machen.

Sprint 3

(30.11.2024 - 13.12.2024)

Ziele:

- Festlegung eines Projektleiter
- weiteren Branch als Zwischenstufe zwischen Entwicklungsstand & *main* anlegen, um Versionierung zu verfeinern und immer eine funktionierende Version verfügbar zu haben
- weiterführende Einarbeitung in Unity
- grundlegende Struktur des Projekts in Unity erstellen
- Erstellung des Projekts auf GitHub
- Steuerungsmöglichkeit des Spielers
- Dungeongenerierung mit vorgefertigten Räumen

Ergebnisse:

Ein Projektleiter wurde gefunden, der die Kommunikation nach außen hin durchführt und innerhalb der Gruppe Termine organisiert.

Es wurde ein neuer Branch *development* angelegt, welcher mit einer Pflicht zum Erstellen eines Pull Requests ausgestattet wurde. Diese Pull Requests müssen genehmigt werden, um auf den *development* Branch Änderungen hinzufügen zu können.

Das Projekt wurde in Unity erstellt und auf GitHub angelegt. Ebenfalls wurden die Steuerungsmöglichkeiten für den Spieler eingebaut, sodass er zwischen einer Top-Down-Perspektive und einer Third-Person-Ansicht wechseln konnte. Es wurde auch ermöglicht, den Spieler mittels der Tasten W, A, S & D zu bewegen und mit Leertaste Projekte abfeuern.

Sprint 4

(14.12.2024 - 27.12.2024)

Ziele:

- Aufgabenerstellung mit Beschreibung, Priorität und Größe
- Zuweisung der Aufgaben und Abarbeitung dieser kleineren einführenden Aufgaben
 - Erstellung eines Canvas, um Nutzer Zahlen schreiben lassen zu können
 - Erkennung von handgeschriebenen Zahlen mit Hilfe von KI
 - Einfaches Lebenspunkte-System mit Darstellung der Lebenspunkte des Spielers
 - Zufällige Platzierung von Gegenständen
 - Korrektur des Verhaltens der schießbaren Projekte des Spielers
 - Einfaches Inventarsystem
 - Aufsammeln von Gegenständen
 - Einführung von Raumtypen

Ergebnisse:

Wir hatten zu Beginn des Sprints klar definierte Aufgaben erstellt und diese zugewiesen an entsprechende Personen.

Es wurde ein Canvas zum Schreiben von Zahlen eingefügt und diese konnte mit einer KI ausgelesen und geprüft werden.

Ein rudimentäres Lebenspunkte-System hat Einzug in das Projekt gefunden mit Hilfe einer Managerklasse. Man hat die Anzahl der Lebenspunkte mit einer Zahl in der Konsole lesen können.

Es wurden die Raumtypen Enemy, Normal, Items, Minigame und Boss eingeführt.

Die Gegenstände wurden in einem Raster zufällig innerhalb eines Raums vom Items Typ platziert, hatten eine zufällige Rotation um die Y-Achse und konnten aufgehoben werden.

Das Inventarsystem wurde eingebaut und konnte mit der I-Taste geöffnet werden.

Eine Korrektur am Projektilverhalten hat stattgefunden, sodass diese immer geradeaus in Blickrichtung des Spielers fliegen.

Sprint 5

(28.12.2024 - 10.01.2025)

Ziele:

- Erstellung weiterführender Aufgaben
 - Gegner fügt Spieler Schaden zu
 - Spieler fügt Gegner Schaden zu
 - Erstellung von Raumvorlagen
 - Verwalten des aktuellen Raumes
 - Entfernen von Gegenständen aus dem Inventar
 - Verwenden eines Gegenstandes hinsichtlich zukünftig geplanter Gegenstände wie Heilungstränke
 - Interactable Interface erweitern
 - wiederholbare Interaktionen
 - wenn man Interaktionstrigger-Bereich verlässt
- Zuweisung der weiterführenden Aufgaben
- Diskussion über die Verwendung von Raycasts mit anschließender Entscheidung

Ergebnisse:

Wenn ein Gegner mit dem Spieler kollidiert, wird diesem Schaden zugefügt und somit seine Lebenspunkte reduziert. Sollte ein Spieler einen Gegner mit seinen Projektilen treffen, so erleiden diese ebenfalls Schaden und werden besiegt und aus der Szene entfernt.

Es wurden Raumvorlagen, orientiert an ihren Typen, mit Hilfe von ScriptableObjects hinzugefügt.

Im *GameManager* Skript wurde der aktuelle Raum gespeichert, dargestellt und sogar nachverfolgt, in welcher Art von Raum der Spieler gerade ist.

Der Spieler kann nun aus dem Inventar Gegenstände entfernen, die final entfernt sind, und kann Gegenstände innerhalb seines Inventars benutzen auf rudimentärer Ebene.

Das Interface *IInteractable* hat die Methoden *OnExit* und *ShouldRepeat* bekommen, um Interaktionen wiederholbar machen zu können und Verhalten abilden zu können, das geschehen soll beim Verlassen des Triggerbereichs des Interaktionselements.

Sprint 6

(11.01.2025 - 07.02.2025)

Ziele:

- Interaktions-Hinweis in der Nähe des Canvas
- Tür-Aktion abhängig vom Minispiel-Status
- Aktualisierung des Frameworks für die Erstellung von Benutzeroberflächen
- Aufsammeln von Items mit einer Anzahl von 0
- Erstellen von Item Vorlagen
- Zufällige Auswahl eines Gegenstandes aus Liste

Ergebnisse:

Wenn der Spieler in die Nähe eines Minispiel-Canvas kommt, bekommt er einen Hinweis, dass er die G-Taste drücken muss, um damit zu interagieren und eine Kamerafahrt zum Minispiel Canvas auszulösen.

Betrifft man einen Minispielraum werden die Türen geschlossen und erst geöffnet bei erfolgreicher Absolvierung dieser.

Das Framework zur Erstellung von Benutzeroberflächen wurde von Unitys eigenem Framework auf UI Toolkit umgestellt.

Es wird verhindert, dass Items mit einer Menge von 0 aufgehoben werden können.

Es sind Itemvorlagenobjekte erstellt worden, die Anzeigebilder, Wahrscheinlichkeiten und geometrische Primitive bekommen haben zur einfachen Darstellung.

Eine Verteiler-Klasse wurde erstellt, die eine Liste von Objekten beliebigen Typs entgegennimmt, deren Erscheinungswahrscheinlichkeiten normalisiert und bei Aufrufen der Methode GetRandomElement einen bestimmten Gegenstand zufällig zurückgibt.

Aufgrund der anstehenden Klausurphase haben sich die Gruppenmitglieder dazu entschieden, sich einmal pro Monat zu treffen.

Sprint 7

(08.02.2025 - 07.03.2025)

Wir mussten uns neu organisieren, weil uns ein Gruppenmitglied auf eigenen Wunsch hin verlassen hat. Daraufhin haben wir diesen Sprint dazu genutzt, ein Lastenheft zu schreiben, um für uns klare und messbare Ziele zu formulieren.

Sprint 8

(08.03.2025 - 03.04.2025)

Ziel dieses Sprints war es, gemeinsam die Aufgaben zu besprechen, zu beschreiben und zu verteilen an die jeweiligen Gruppenmitglieder. Dies konnten wir zum Ende des Sprints als Ergebnisse vorweisen.

Sprint 9

(04.04.2025 - 17.04.2025)

Ziele:

- Erstellung der Aufgaben aus Lastenheft
- Versehung der Aufgaben mit Priorität und geschätzter Größe
- Einteilung der Gruppenmitglieder in primäre Aufgabengebiete
- Initiale Diskussion zu Speicherung des Spielzustandes

Ergebnisse:

Alle Aufgaben, die im Lastenheft aufgeführt wurden, wurden ins Kanban Board gepflegt mit Priorität und geschätzter Größe.

Es wurde eine initiale Diskussion darüber geführt, wie man unser Spiel speichern kann, um Fortschritt zu speichern. Dabei ist aufgefallen, dass wir einen Seed brauchen, um bei unseren randomisierten Vorgehen Determinismus zu gewährleisten.

Sprint 10

(18.04.2025 - 08.05.2025)

Ziele:

- Verschiedene Shader einbauen um visuelle Erscheinung zu verbessern
- Einbindung erster verschiedener Gegnertypen
- Erstellung des Dungeons nach dem Prinzip eines Voronoi-Diagramms

Ergebnisse:

Wellen, Vertex-Displacement und Voronoi Shader wurden in das Projekt eingefügt und für die spätere Verwendung vorbereitet.

Das Paket ML-Agents wurde in das Projekt eingebunden und in einer Testszene verwendet mit zwei verschiedenen Gegnertypen, welche beide den Spieler verfolgen. Einer simpel, ohne Hindernisse umgehen zu können und der andere kann Hindernisse überwinden. Dritter Gegnertyp ist in Arbeit

Die Grundlegende Struktur der neuen Dungeon-Generierung wurde gelegt, welche noch kleine Unregelmäßigkeiten aufweist in der Kantengenerierung.

Sprint 11

(09.05.2025 - 21.05.2025)

Ziele:

- Einbauen eines Pausemenüfensters
- Einbauen eines Startbildschirms für das Spiel
- Korrektur der Dungeongenerierung
- Setzen von Türen in Dungeonmauern
- Erhöhung der Wartbarkeit des Codes der Dungeongenerierung
- Speicherung des Seeds für die Initialisierung bei Verwendung von zufälligen Werten

Ergebnisse:

Um das Spiel zu speichern, einen bisherigen Spielstand zu laden oder das Spiel sogar zu verlassen, wurde ein Pausemenüfenster implementiert. Hierfür wurde eine Klasse *UIManager* erstellt, welcher die generelle 2D-Benutzeroberfläche steuern soll. Hinzu kam ein *ButtonManager* welcher die Buttoninteraktionen mit dem Pausemenü ermöglicht, aber noch keine Auswirkungen hatte, außer Konsolenausgaben zu senden. Der *UIManager* hat auch die Eingabe des Spielers unterbunden, sodass man nicht die Spielszene beeinflusst, während man im Pausemenü ist.

In einer neuen Szene wurde ein Startbildschirm eingebaut, welcher die Spielszene laden soll, sobald auf den entsprechenden Button geklickt wird.

Es ist während der Entwicklung der Dungeongenerierung aufgefallen, sollten Umkreismittelpunkte der Dreiecke im Voronoi-Diagramm außerhalb der Mauern der Grenzen erzeugt werden, so wurden die Wände länger gezogen, als notwendig. Dieses Verhalten wurde korrigiert.

Damit das Vorankommen des Spielers gewährleistet wird in der neuen Variante der Dungeongenerierung, werden Türen in der Mitte der gezogenen Mauern gesetzt.

Um der erschwerten Wartbarkeit des Codes der Dungeongenerierung entgegenzuwirken, wurde das Prinzip ein wenig abstrahiert und eine Datei mit dem Namen *DungeonGraph* erstellt, welche einen Graph anhand der Punkte des Voronoi Diagramms erstellt und somit die Berechnung des Diagramms an sich ausgelagert wurde.

Für die Reproduzierbarkeit von der Erstellung des Dungeons und anderer Generierungen mithilfe von zufälligen Werten wurde ein globaler Wert angelegt und gespeichert, der die Zufallsverteilung initialisiert.

Sprint 12

(22.05.2025 - 05.06.2025)

Ziele:

- Visuelle Darstellung der Gegner
- Verschiedene Varianten von Gegnern
- Hinzufügen eines weiteren Minispiele
- Nachverfolgung von besuchten Räumen bei neuer Dungeongenerierung
- Türschließmechanismus für Raumtypen
- Überarbeitung Inventar Benutzeroberfläche
- Ausrüstung hinzugefügt
- Hinzufügen von Spitzhacke als Gegenstand
- Steuerung der Raumzahl

Ergebnisse:

Die Gegner haben abhängig von ihren Typen entsprechende 3D-Modelle bekommen, um diese besser unterscheiden zu können, wenn diese verwendet werden.

Es wurden neue Varianten von Gegnertypen eingebunden, Big Stalker und Shooter.

Die Glyphenerkennung wurde als ein weiteres Minispiel eingebunden. Hier geht es darum, eine Glyphe auf ein Canvas zu zeichnen und von einer KI einschätzen zu lassen, ob diese korrekt wiedergegeben wurde.

Um Events auslösen zu können, abhängig vom Raum, muss der aktuelle Raum, in welchem sich der Spieler befindet, nachverfolgt werden, dies wurde nun auch mit der neuen Dungeongenerierung umgesetzt anhand der Distanz zu den zufällig generierten Voronoipunkten.

Ebenfalls wurde aufgrund der neuen Dungeongenerierung der Türschließmechanismus auf entsprechende Raumtypen angewandt und aktualisiert.

Da die Möglichkeit Gegenstände auszurüsten und somit Ausrüstung möglich gemacht wurde, lag es nahe die Inventar Benutzeroberfläche zu überarbeiten, um dort auch die Ausrüstung darstellen zu können.

Es wurde die Spitzhacke zum einreißen von Mauern hinzugefügt und über den Weg mithilfe eines minimalen Spannbaums im nächstgelegenen Itemraum instanziert.

Sprint 13

(06.06.2025 - 15.06.2025)

Ziele:

- Gegner sind besiegtbar
- Weitere Gegnertypen hinzufügen
- Wände zerstörbar machen
- Bossraum Türen bleiben geschlossen
- ItemInstance als abstract ScriptableObject

Ergebnisse:

Da bisher nur Platzhalter in der Schadenslogik verwendet wurden, mussten die neuen Gegner ebenfalls mit der Logik des Schadenerleidens und -zufügens verbunden werden.

Für mehr Variation bei den Gegnern wurde ein Typ hinzugefügt, welcher hinzugefügt, die schwächere Duplikate von sich erzeugen können (Spawner). Ein weiterer neuer Gegnertyp kann sich schnell mit kleineren Teleports in Richtung des Spielers bewegen. Zuletzt wurde eine Variante hinzugefügt, welche sich nur dem Spieler nähert, wenn er diesen auch sieht.

Um den Dungeon individuell erkunden zu können, wurden zerstörbare Wände hinzugefügt, welche mit einer Spitzhacke eingerissen werden können.

Die Funktionalität, dass solange kein Schlüssel zur Tür eines Bossraums ausgegeben wird, die Türen zu diesem Raum geschlossen bleiben, wurde eingebaut.

In unseren Skripten hat sich ergeben, aufgrund der Möglichkeiten hinsichtlich Funktionalitäten und Verwendung, dass wir aus der Klasse *ItemInstance* ein *ScriptableObject* machen, um diese leichter instanziieren und variieren zu können.

Sprint 14

(16.06.2025 - 21.06.2025)

Ziele:

- Instanziierung von Gegenständen in neuer Dungeongenerierung
- Icons für Ausrüstungsplätze

Ergebnisse:

Durch die neue Generierung des Dungeons und dessen Räume, musste die Gegenstandsplatzierung und -instanziierung neu implementiert werden. Man hat in den Räumen die kürzeste Distanz zu den Nachbarn ausgewählt, diese halbiert und mit einem Sicherheitsfaktoren versehen, um zu gewährleisten, dass man sicher im Raum bleibt, wenn man etwas positionieren möchte.

Um zu erkennen, welche Art von Ausrüstung auf welchen Platz gehört, wurden selbst erstellte Platzhalterbilder in das Ausrüstungsinventar eingebaut.

Sprint 15

(22.06.2025 - 28.06.2025)

Ziele:

- Erweiterung der Erscheinungslogik um Gegner- und Minispielinstanziierung
- Finalisierung des Inventarsystems

Ergebnisse:

Die Instanziierungslogik für Objekte wurde nun erweitert, um Gegner und Minispiele zufällig erscheinen zu lassen. Hierfür wurden die Klassen *EnemySpawnerVoronoi* und *MiniGameSpawnerVoronoi* erstellt.

Das Inventarsystem wurde soweit fertiggestellt, dass es verwendbar ist mit den neuen Klassen *Item* und es auch mit Ausrüstung funktioniert. Ausrüstung hat die Klasse *Equipment* bekommen, um die Zuordnung im Inventar zu erleichtern von Programmiererseite.

Sprint 16

(29.06.2025 - 06.07.2025)

Ziele:

- Bossraumschlüssel hinzufügen
- kleinere Codesäuberungen
- Refactoring von *InventoryManager*

Ergebnisse:

Der Bossraumschlüssel wurde ganzheitlich hinzugefügt, das bedeutet, dass sowohl 3D-Modell, ein Bild für das Inventar und das entsprechende Skript implementiert wurden.

Es haben auch kleinere Codesäuberungen stattgefunden, so wurde aus der Klasse *Inventory_V3* die Klasse *Inventory* im gesamten Projekt oder auch an diversen Stellen Kommentare hinzugefügt.

Im gleichen Zuge wurde das Skript des *InventoryManager* verbessert, sodass er jetzt bei seinen Membervariablen Kommentare hat, wofür diese sind, die Funktionen wurden ebenfalls kommentiert und diverse kleinere Funktionen hinzugefügt, um die Les- und Wartbarkeit zu erhöhen.

Sprint 17

(07.07.2025 - 13.07.2025)

Ziele:

- Lebenspunkttestand visuell untermalen mit Vignette
- Lebenspunktebalken einbauen
- Dungeonkarte einbauen
- Bossraumtür erkennbar machen

- Spitzhacke visuelle und codeseitige Grundlage
- Anpassen der Säulentrefferbox

Ergebnisse:

Der Lebenspunkttestand des Spielers wurde visuell untermauert mit Hilfe einer Vignette, die auf das Bild gelegt wird und je nach Lebenspunkttestand etwas mehr Bedrohlichkeit der Situation darstellt.

Ein Lebenspunktebalken wurde in der generellen Benutzeroberfläche hinzugefügt, damit der Spieler sehen kann, wie viele Lebenspunkte er im Verhältnis zum maximalen Leben hat.

Es wurde eine Karte zur Orientierung im Dungeon eingebaut, die man durch Drücken der M-Taste öffnen kann und immer in die Blickrichtung des Spielers mittels eines Pfeils darstellt.

Die Bossraumtüren wurden mit einem Shader versehen, sodass erkennbar ist, dass sich diese Türen von den anderen aus dem Dungeon unterscheiden und diese nicht einfach geöffnet werden können.

Einer der Schlüsselgegenstände, die Spitzhacke, haben mit einem 3D-Modell, Bild für das Inventar und Anlegen als *ScriptableObject* Einzug ins Projekt gefunden.

Beim Testen des Spiels ist aufgefallen, dass manchmal der Durchgang durch eine Tür verhindert wird. Das lag daran, dass die Türen sehr nah an Säulen waren, welche eine größere Kollisionsbox hatten, als notwendig und somit den Spieler daran hinderten, die Tür durchqueren zu können.

Sprint 18

(14.07.2025 - 20.07.2025)

Ziele:

- Spitzhacken Interaktion mit zerstörbaren Wänden
- Statuswerte einbauen

Ergebnisse:

Nachdem die Spitzhacke im vorherigen Sprint ins Projekt gekommen ist, wurde in diesem Sprint die Interaktion mit den zerstörbaren Wänden eingebaut, sodass man sich nun durch den Dungeon auch durch diese Wände bewegen kann.

Es wurde auch eine Klasse implementiert, die Statuswerte speichert und diese aktualisiert.

Sprint 19

(21.07.2025 - 25.07.2025)

Ziele:

- Gegenstand kenntlich machen mit Hilfe eines Shaders um 3D-Objekt in der Szene
- Statuswerte global verwendbar machen
- Entfernen alter Lebenspunkteklassen

Ergebnisse:

Nachdem Gegenstände manchmal nicht erkennbar waren aufgrund ihrer 3D-Modelle, wurden diese um einen Shader erweitert.

Das Statuswert Objekt wurde vom Spieler auch auf Gegner ausgeweitet, um die Werte besser steuern und ansprechen zu können.

Aus vorherigen Aufgaben waren noch Dateien und Klassen vorhanden, die nicht mehr unseren Programmcode-Ansprüchen genügten und mangelhaft funktional waren, wurden diese aufgrund des neu eingeführten Statuswertobjekts hinausgeworfen.

Sprint 20

(26.07.2025 - 03.08.2025)

Ziele:

- In *VoronoiGenerator* den globalen Initierungswert für Zufallswerte verwenden
- Seltenheitsfarbe des Gegenstandes bei Shader in 3D-Szene verwenden
- neue Vorlage für Projektile
- Einbauen von unterschiedlichen Skyboxen
- Anpassung des globalen Lichts an Shader des Bodens
- Grundlegende Struktur der Speicherliste

Ergebnisse:

In der Klasse *VoronoiGenerator* wurden die sehr früh angelegten Shader verwendet zur Veränderung der Darstellung des Bodens und um auch die Erscheinung dieser deterministisch zu machen, musste der Initierungswert auch hier Einzug finden.

Da die Gegenstände unterschiedliche Seltenheiten haben und das im Vorfeld nicht erkennbar war, welche Gegenstände selten sind und welche nicht, wurde die Shaderfarbe an die Farbe der Seltenheit für das Inventar angepasst.

Dadurch, dass die Gegner, als auch der Spieler das gleiche Projektilobjekt verwenden, wurde eine neue Vorlage für die Projektile der Gegner erstellt, um diese besser unterscheiden zu können.

Um die Stimmigkeit der visuellen Darstellung ein wenig anzugeleichen, wurden neue Skyboxen durch das Paket von Boxophobic hinzugefügt und farblich abgestimmt.

Das globale Licht innerhalb des Dungeons wurde auch durch das Skript angesteuert und entsprechend farblich an die Shader des Dungeons angepasst, um ein visuell ansprechendes Erlebnis zu schaffen.

Das Thema Speicherstand stand schon länger auf der Liste der Themen, die wir besprechen und angehen mussten und haben es angegangen und uns darauf geeinigt, welche Sachen definitiv gespeichert werden müssen, welche vernachlässigbar sind. Somit haben wir uns auf eine grundlegende Struktur geeinigt, die Umsetzung erfolgt im nächsten Sprint.

Sprint 21

(04.08.2025 - 17.08.2025)

Ziel dieses Sprints war es, sich ausschließlich auf das Speichersystem zu fokussieren. Dadurch, dass an vielen Stellen Zufallswerte verwendet wurden und das ein ziemlich komplexes Thema ist, wollten wir das als Gruppe in gemeinsamen Meetings angehen.

In den nachfolgenden Sprints ging es darum, weitere Fehlverhalten zu beheben, da die Kernfunktionalitäten, bis auf wenige Ausnahmen, bereits vorhanden sind.

Sprint 22

(18.08.2025 - 24.08.2025)

Ziel dieses Sprints ist es möglichst viele unerwünschte Verhaltensweisen und Bugs zu finden und diese zu beheben, die uns während des Spielens aufgefallen sind. Ebenfalls wollen wir die letzten fehlenden Aufgaben abschließen.

Ergebnisse:

Da die Türen in manchen Situationen bis zu einem gewissen Grad skaliert werden in ihrer Breite, hatte dies zur Folge, dass der Spieler diese nicht durchqueren konnte. Zur Lösung dieses Problems haben wir uns dazu entschieden, die Kollisionsbox des Spielers zu verkleinern und mehr an die Größe der Figur anzupassen.

Es gibt Gegenstände, die instanziert werden müssen und diese sollten exakt einmal vorhanden sein auf der Ebene, alle Itemräume übergreifend. Diese Funktion haben wir eingeführt, indem wir bei der Verteilung für die *Distributor* eine weitere Liste eingefügt haben, die Pflichtgegenstände darstellt und diese zuerst zurückgibt, bevor ein zufälliges Element ausgewählt wird.

Um das Leben des Spielers wieder auffüllen zu können, haben wir einen Lebenspunktetank in das Spiel eingefügt und zu den Elementen hinzugefügt, die im Dungeon erscheinen können.

Sprint 23

(25.08.2025 - 03.09.2025)

Da wir mit den offenen Punkten unserer Liste im letzten Sprint nicht fertig geworden sind, haben wir in diesem Sprint mit der Abarbeitung dieser weitergemacht. Entsprechend gestaltet sich unser Ziel, in diesem Sprint diese vollständig abzuarbeiten.

Ergebnisse:

Bis zu diesem Sprint war noch kein Verhalten implementiert, sollte die Lebenspunkte des Spieler auf 0 sinken. Wir haben somit ein Fenster eingebaut, welches anzeigt, dass das Spiel vorbei ist, da man gestorben ist. Im Anschluss hat man die Möglichkeit, seinen letzten Speicherpunkt erneut zu laden, ein neues Spiel zu starten oder das Spiel zu verlassen.

Da die Türen verschlossen werden, sollte man in einen Fragenminispielraum kommen, öffnen sich diese nun nach erfolgreichem Absolvieren der Fragen.

Der Bossraumschlüssel wird nun ausgehändigt, nachdem das Minispiel zum Nachzeichnen der Glyphen erfolgreich abgeschlossen wurde.

Die Größe und Startposition zum Boden wurde an den Gegner angepasst, um ein stimmigeres Erlebnis zu schaffen. Im Zuge dessen wurde auch das Statuswertobjekt bei den Gegner angebunden, sodass alle Werte, wie Geschwindigkeit, Leben und Schaden einheitlich gesteuert werden können. Die Statuswerte wurden im Anschluss auch angepasst, sodass das Spielerlebnis besser wurde.

Die Türen in einem Gegnerraum schließen sich nach Betreten und öffnen sich nun erst, wenn alle Gegner besiegt sind.

Im Rahmen des Testens ist uns aufgefallen, dass der Effekt der Spitzhacke eine Mauer zerschlagen nicht an den Ablauf der Animation gekoppelt war, dies haben wir ebenfalls behoben und die Mauer kann nicht weniger als 0 Lebenspunkte haben (@Todo: check wie viel Leben die Mauer hat).

Um im Bossraum etwas anspruchsvollere Gegner anzutreffen, wurde eine Liste von Gegnern für den Bossraum erstellt und ebenfalls Hindernisse hinzugefügt, um einen bisher noch gar nicht genutzten Gegnertypen volumnäßig einsetzen zu können.

Es wurden 3D Modelle aus einem externen Paket importiert, um ein paar Gegenstände visuell ansprechender zu gestalten.

Die Vignette für die Untermalung der Lebenspunkte hatte zuvor nur bei der Reduzierung von Lebenspunkten einen linear interpolierten Übergang gemacht, dies ist nun auch bei der Erhöhung von Lebenspunkten der Fall.

Zur Bestimmung der zu zeichnenden Glyphe wurde ein sammelbarer Gegenstand in das Spiel eingebunden, der die vorgesehene Glyphe dem Spieler zeigt.

Nachdem das Statuswertobjekt sowohl beim Spieler, als auch bei Gegnern eingeführt wurde, ist uns aufgefallen, dass es einige Unregelmäßigkeiten in der Lebenspunkteberechnung gab, wenn ein Gegner mit dem Spieler kollidiert. Daraufhin haben wir die Interaktionsskripte *PlayerInteraction* und *EnemyInteraction* korrigieren müssen, damit die Werte korrekt aktualisiert werden.

Sprint 24

(04.09.2025 - 10.09.2025)

Ziele:

- Hinzufügen von Vegetation in Dungeon
- Levefortschritt hinzufügen
- Korrektur der Spielerfigur
- Hinzufügen verschiedener Heilungstränke
- Koppelung von tatsächlicher und hinterlegter Spielergeschwindigkeit
- Skalierung von Werten anhand des Levels
- Hinzufügen von Levefortschrittanzeige
- Ausschluss von Überlagerung mehrerer Benutzeroberflächen
- Korrektur von kürzester Weg zu Gegenstandsraum
- Abspeichern des Fortschritts von Minispiele

Ergebnisse:

Da der Dungeon an einigen Stellen etwas leer ausgesehen hat, haben wir uns dazu entschieden, die Darstellung etwas zu verbessern, indem Pflanzen im Dungeon hinzugefügt werden.

Um ein Gefühl von Belohnung beim Spieler zu erzeugen, haben wir in diesem Sprint den Levefortschritt hinzugefügt und ebenfalls eine Anzeige in den Pause- und Spielendebildschirm dafür eingebaut, damit ersichtlich ist, in welchem Level man sich befindet beziehungsweise wie weit man gekommen ist. Gleichzeitig haben wir auch eine Skalierung von Werten bei Gegnern und Bossen hinzugefügt, damit sich das Fortschreiten auch umfassender anfühlt.

Da die Spielerfigur immer noch etwas unpassend war hinsichtlich der Breite, wurde dies korrigiert.

Eintrag haben verschiedene Lebenspunktetränke in das Spiel gefunden, welche 25%, 50%, 75% oder 100% der maximalen Lebenspunkte heilen.

Bisher waren die Bewegungsgeschwindigkeit des Spielers und die eingetragene Geschwindigkeit des Spielers aus seinem zugehörigen Statuswertobjekt nicht verbunden, was bei der Ausrüstung von Gegenständen, welche die Geschwindigkeit beeinflussen, dafür sorgte, dass sich nichts verändert hatte, dieses Unstimmigkeit wurde behoben.

Bei der Skalierung von Werten anhand des Levels wurden nicht nur die Gegner und Bosse berücksichtigt, sondern auch die Größe des Dungeons und dessen Raumzahl, welche sich nun mit dem Fortschreiten des Levels erhöht.

Sobald eine Benutzeroberfläche aktiviert wurde, sei es die vom Inventar oder vom Pausemenü, so konnten die jeweils nicht aktiven Benutzeroberflächen noch aktiviert werden und überlagerten die zuvor aktivierten. Dies ist gelöst worden, indem mehr Abfragen in den *UIManager* eingebaut wurden beim Drücken der jeweiligen Tasten zum zugehörigen Bildschirm.

Der kürzeste Weg zum nächstgelegenen Itemraum, um dort die Spitzhacke erscheinen zu lassen, führte manchmal über einen Bossraum, was uns dazu zwang die Methode *FindShortestPathWithoutBossRoom* einzubauen, damit die Spitzhacke immer erreichbar ist.

Da die Minispiele bei speichern innerhalb eines Levels neu gespielt werden mussten, haben wir sie daraufhin in unsere Liste der zu speichernden Zustände aufgenommen, damit der Spieler nicht beim Laden eines Spielstandes dazu gezwungen wird Minispiele erneut absolvieren zu müssen.

Sprint 25 (finaler Sprint)

(11.09.2025 - 25.09.2025)

Ziele:

- Zusammenführung der Dokumentation des Codes
- Entfernen von nicht mehr notwendigen Paketen und Codeteilen

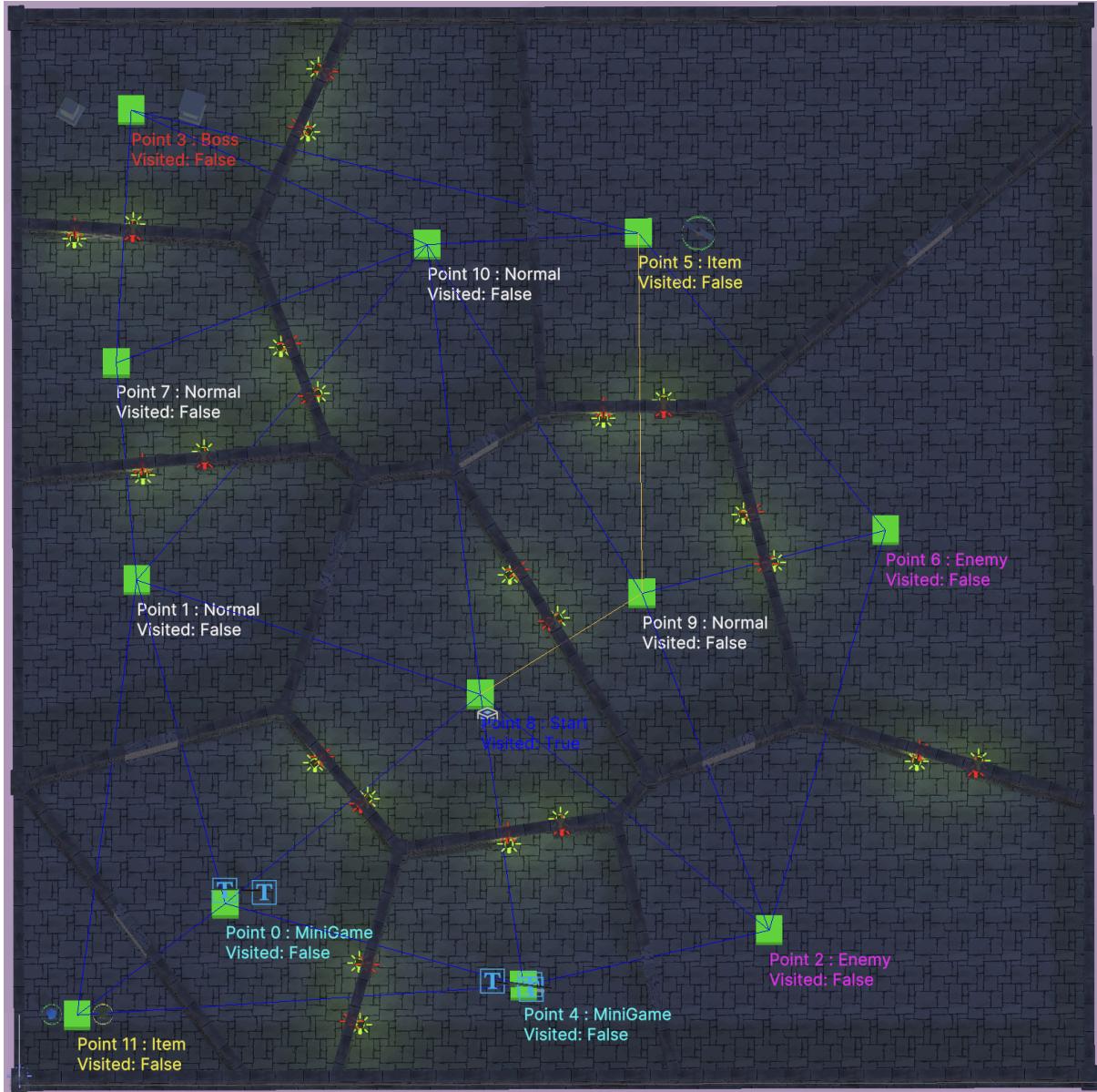
Ergebnisse:

Der Programmcode wurde im vorherigen Sprint mit Kommentaren versehen. In diesem Sprint war die Zusammenführung davon notwendig, da es in manchen Dateien Veränderungen von zwei oder mehr Personen gab und dies mit Vorsicht zu lösen gilt.

Im Anschluss an die Zusammenführung wurden Codeteile und eingebaute Pakete entfernt, die nicht mehr Verwendung im Projekt gefunden haben. Dies erleichtert die Übersicht im Projekt zu bewahren.

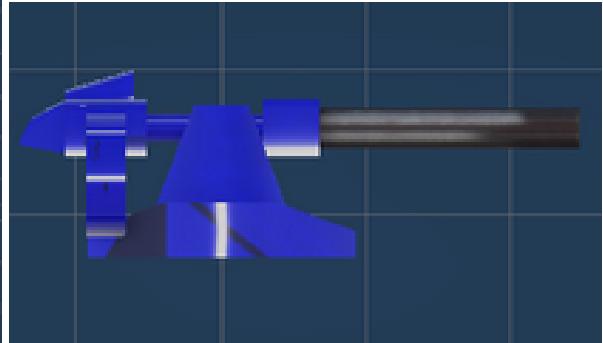
Mediengalerie

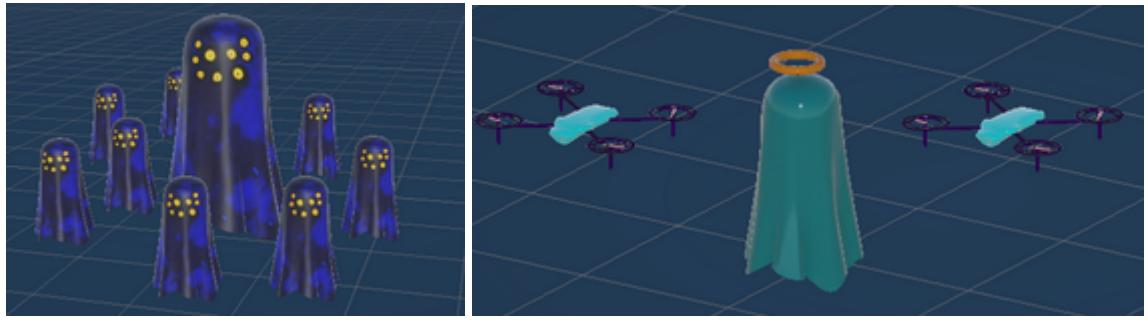
Dungeon-Ansicht





Kämpfe & Gegner



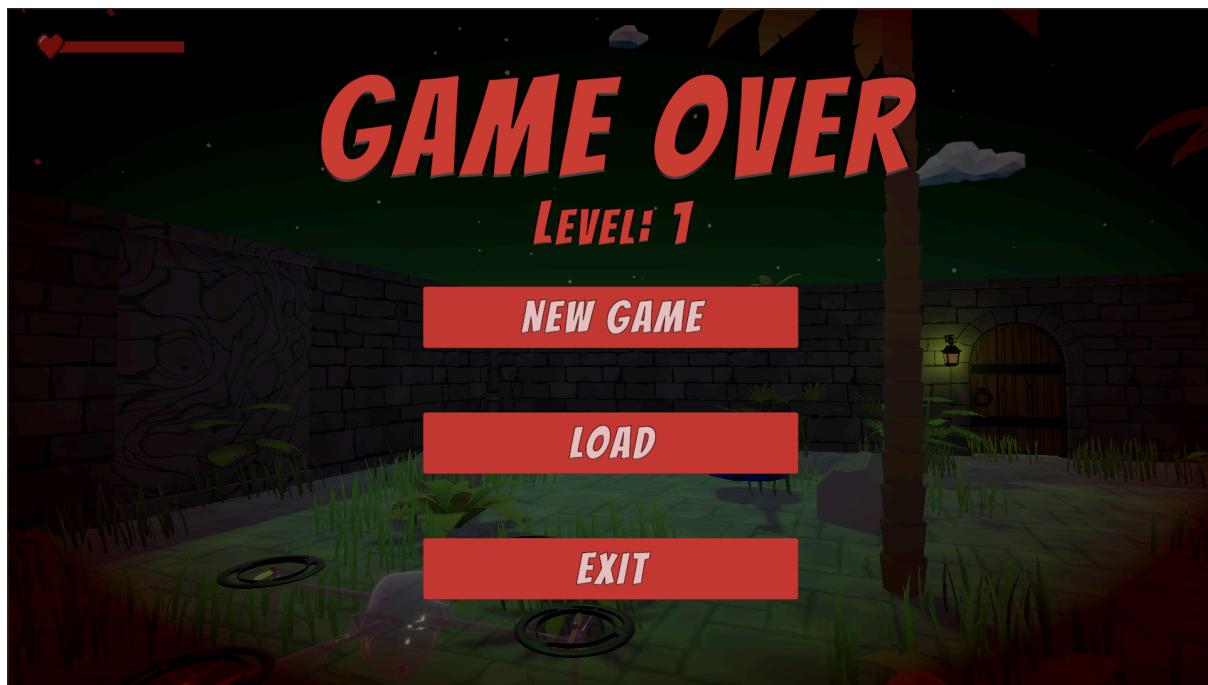


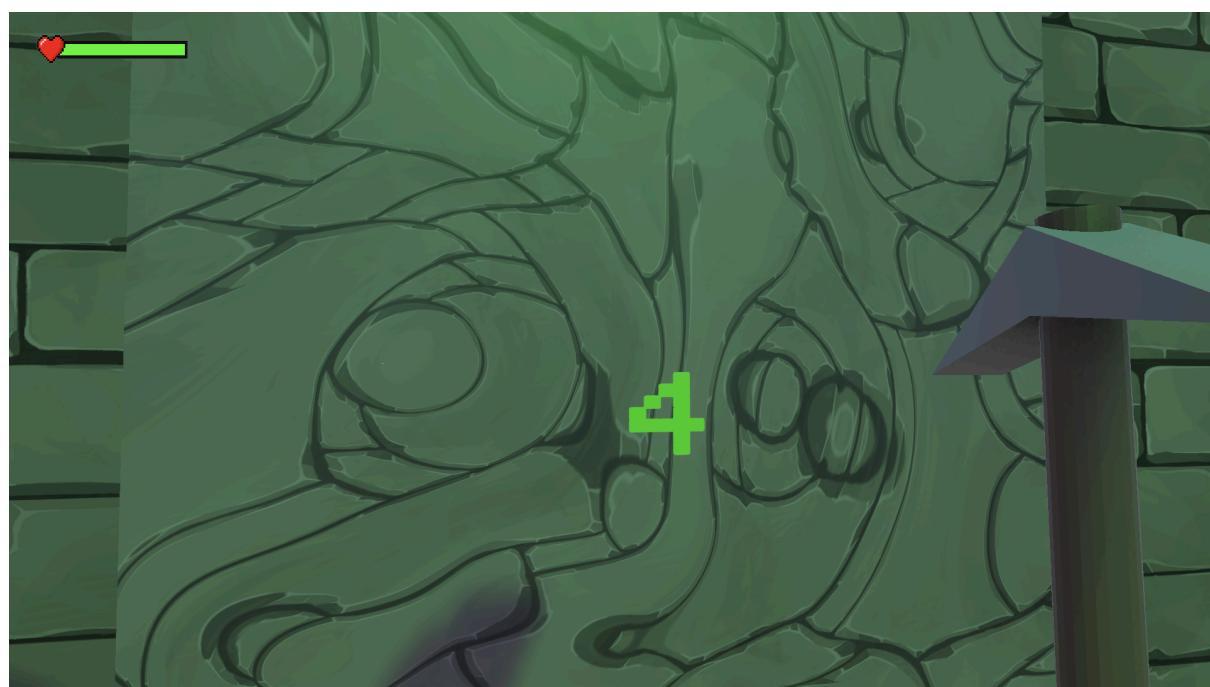
Rätselräume / Minispiele



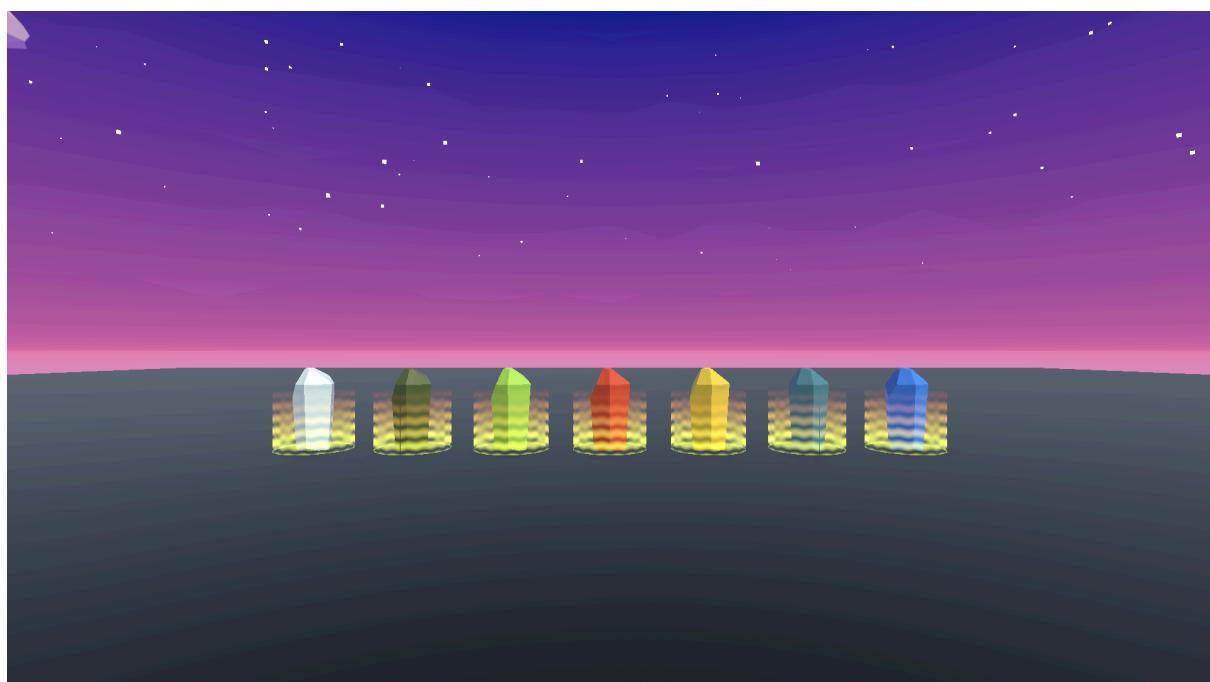
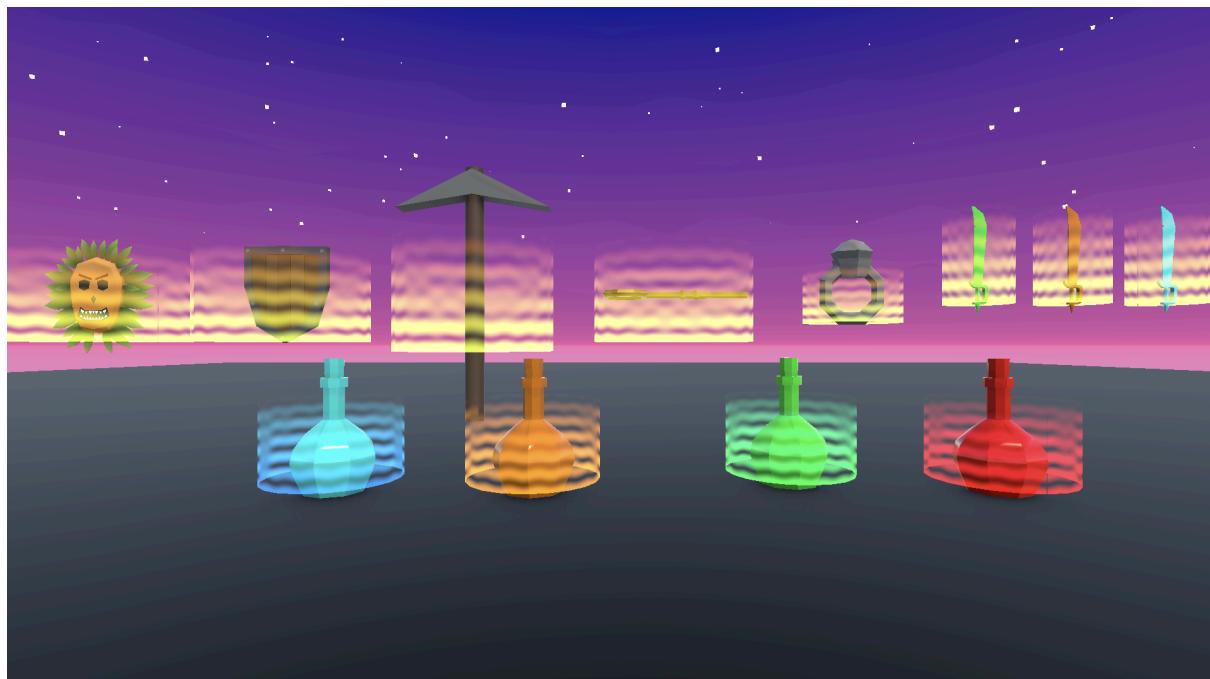
UI&Inventar







Items



Shader

