Mendizils Vermächtnis

C#Konsolenspiel Dokumentation

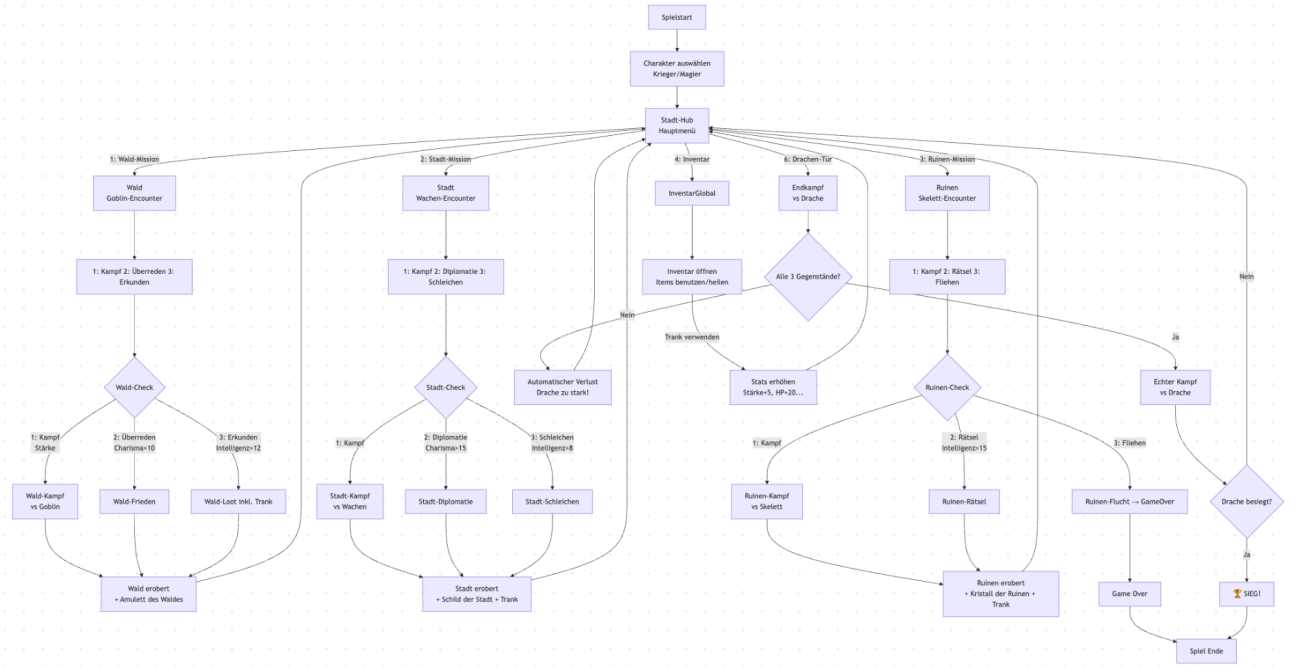
Nicholas Kubbutat

Inhaltsverzeichnis

Projektantrag

Die ursprüngliche Idee für das Spiel war ein schlankes Quest-Gerüst mit klaren Rollen und drei Lösungswegen. Nach der Charakterauswahl (Kämpfer, Barde oder Schurke) sollte der Spieler in einer Stadt als Hauptmenü landen, dort einen Begrüßungstext lesen und anschließend zwischen drei Missionen, dem Inventar und einem Drachenkampf wählen. Direkt nach der Missionswahl war vorgesehen, den Status zu prüfen: Ist die Quest bereits erledigt, erhält der Spieler nur eine kurze Rückmeldung wie „Alles friedlich“ und kehrt sofort in die Stadt zurück.

Für die drei Missionen – Wald, Stadt und Ruinen – war ein einheitlicher Ablauf geplant: Zuerst ein Status-Check, dann ein Encounter mit den Optionen Kampf, Diplomatie oder Schleichen. Der Kampf sollte als Auto-Battle funktionieren, bei dem zufällig Treffer bestimmt werden und ein Verlust der Lebenspunkte auf 0 unmittelbar zum Game Over führt.



Flowchart Projektantrag

Bei Misslingen von Schleichen oder Diplomatie, kehrt der Spieler ohne Fortschritt einfach in die Stadt zurück. Gelingt der Encounter – unabhängig vom gewählten Lösungsweg – erhält der Spieler ein Quest-Item und die jeweilige Mission wird dauerhaft als „erledigt“ markiert.

Beim Drachen war vorgesehen, dass ein Versuch ohne alle drei Quest-Items automatisch scheitert und der Spieler in die Stadt zurückgesetzt wird, während der Kampf mit allen Items als finaler Bosskampf angelegt ist und bei Sieg das Spiel beendet.

Darüber hinaus waren einige optionale Erweiterungen vorgesehen, falls die Zeit es erlaubt: Heilungsmöglichkeiten in der Stadt (z.B. Tempel oder Bar), zusätzliche oder aufgewertete Items sowie spannendere Mechaniken für Schleichen und Diplomatie, etwa direkte Kopplung an die Charakterwerte oder an bestimmte gefundene Gegenstände.

Projektablauf

1.12. – Projektstart und Grundgerüst

- Projekt initialisiert und erstes Konsolenspiel-Projekt angelegt („Projektinitialisierung“).
- Charakter -Klasse erstellt, um gemeinsame Werte für Spieler und Gegner zu modellieren (HP, Stärke, etc.).
- Hilfsmethoden -Klasse angelegt mit Begrüßung und Charakterwahl (Spielerklassenauswahl + Name).
- Item und Items eingeführt sowie im Spieler eine Inventar-Liste und Methode InventarAnzeigen() hinzugefügt, um Items strukturiert verwalten zu können. (War unnötig…)

2.12. – Encounters und Stadthub

- Encounter.cs und Waldencounter.cs hinzugefügt – erster echter Encounter (Wald/Goblin)[1]
- Ein grober Stadthub erstellt, von dem aus der Spieler in unterschiedliche Bereiche (z.B. Wald) wechseln kann.

3.12. – Encounter-Rework und neue Orte

- Kampfmethode, Sneak-Methode usw. aus den einzelnen Encounter-Klassen in die allgemeine Encounter -Basisklasse verschoben, um Code-Duplikation zu vermeiden.
- Mehrere neue Encounters ergänzt und die Kampfmethode angepasst (z.B. bessere Anzeige, einheitliches Verhalten).
- Branch „Encounter-Rework“ mit main gemerged; zentrale Encounter-Logik nun in der Basisklasse .
- Option „Portrait anzeigen“ zu Encountern hinzugefügt (ASCII-Gesichter).
- Drachenencounter.cs hinzugefügt – Grundstein für die finale Boss-Quest.
- Item-Abfrage vor dem Drachen-Encounter eingebaut, sodass die Drachenquest nur startet, wenn der Spieler die nötigen Artefakte besitzt.
- StadtErkunden.cs erstellt mit Platzhaltern für Tempel, Bar und weitere Stadt-Aktivitäten.

4.12. – Quests, Gerüchte und Kampfsystem

- Quest-Logik aus Program.cs in eine neue Quests.cs ausgelagert, damit jede Quest als eigene Methode verwaltet wird.
- Ein Questboard eingeführt und aus Program in Hilfsmethoden ausgelagert, um die Hauptschleife zu vereinfachen.
- Nicht mehr benötigte, abgeleitete Gegnerklassen gelöscht; Gegner werden nun direkt im jeweiligen Encounter erzeugt.(Das war ein Fehler)
- Neue Klasse Gerüchte.cs erstellt, inklusive einer Liste von Gerüchten und einer Methode „Gerüchte hören“.
- Anpassungen an Gerüchte.cs , um bekannte Gerüchte im Spieler zu speichern und nur neue Gerüchte anzuzeigen.
- Verschiedene Story-Anpassungen vorgenommen (Dialoge, Lore zu Heldin, Drache, Magister).
- Neues Kampfsystem eingebaut: Umstellung von statischem Schaden auf eine DnD-ähnliche Würfellogik mit W20, Treffer-Schwelle und Krits.
- Balancing wurde angepasst

[1] LLM Nutzung: Komplett von AI gecoded Im “Lehrermodus” Texterstellung

5.12. – Quest-Fortschritt und Bar-Vorbereitung

- Neue Gegner.cs wieder eingeführt, um spezielle Fail-Texte (SneakFail ,KampfFail , ÜberredenFail) zentral pro Gegner zu verwalten
- Quests.cs erweitert, sodass Quests nur einmal erfolgreich abgeschlossen werden können (Abfrage ob Spieler Questerfolgs Item im Inventar hat).
- Encounter.cs angepasst, damit Erfolgsnachrichten sauber nach den Encounter-Texten angezeigt werden.
- BesucheBar angelegt – erste Struktur für den Bar-Besuch mit Platzhaltern für spätere Encounter .

6.& 7.12.– Savegame und fertiger Bar-Encounter

- Save-Funktion hinzugefügt: Spielstand des Spieler wird als JSON gespeichert und kann beim Start geladen werden.
- Quests weiter ausgebaut.
- Tempel und Bar erweitert (Heilung im Tempel, Zimmer in der Bar, Krug bestellen).
- Bar-Encounter fertiggestellt, Kurze Dialoge falls andere Missionsitems im Inventar sind.
- Fehler behoben, bei dem der Bar-Encounter jedes mal ausgelöst wurde.

8.12. - Dokumentation und Finetuning

- Großteil des Tages mit der Projektdokumentation verbracht.
- Dabei gemerkt, dass ich mich viel zu sehr auf die Commit-History verlassen habe.
- Viele Fehler, Probleme und Zwischenschritte musste ich mir rückwirkend zusammenreimen.
- Erkenntnis: Nächstes Mal ab Tag 1 konsequent mitloggen und Doku parallel zum Code schreiben.

9.12- Dokumentation/Polish

- Überreden angepasst, Überreden Fail führt jetzt zu Kampf
- Anpassung Hilfsmethode Weiter(damit man nicht in jeden Encounter ConsoleReadkey() etc. oft verwenden muss
- Spieler Portraits hinzugefügt und Option in der Stadt in Brunnen zu schauen für Porträt
- Drachen encounter fertiggestellt
- Methode Heilen/Schaden in spieler.cs und Anpassungen an stellen wo Spieler geheilt wird
- Item/Items zu einer Klasse gemacht, das hat überhaupt keinen Sinn gemacht..
- Text zu Beginn des Spiels eingefügt
- Texte Angepasst
- Anpassung mit Console.Clear, dass immer wenn möglich nur das Menü oben ist

Umsetzung & Überarbeitungen

Programm.cs

Umsetzung

Beim Start fragt Main zuerst ab, ob ein neues Spiel gestartet oder ein Savegame geladen werden soll. Wählt der Spieler „2“ und es gibt einen gültigen Speicherstand, wird dieser geladen und der Spieler begrüßt, sonst läuft es automatisch auf die Charakterwahl hinaus.
Danach läuft eine While-Schleife, in der über Questboard() die nächste Aktion abgefragt wird: Stadt erkunden, eine der drei Quests (Magister, Ruinen, Wald), die Drachenquest, Inventar anzeigen, speichern oder beenden. Je nach Eingabe wird die passende Quest-Methode(welche dann die Entsprechende Encounter aufruft) mit dem aktuellen Spieler aufgerufen, das Inventar gezeigt, der Spielstand gesichert – oder bei „0“ eine Abschiedsnachricht ausgegeben und spielLaeuft auf false gesetzt, womit das Programm endet.

Überarbeitungen

Zu Beginn war der Plan, die komplette Spiellogik direkt in den switch -Cases der Hauptschleife zu erledigen: Für jede Auswahl sollte dort ein neuer Encounter erzeugt, durchgespielt und danach direkt das entsprechende Item vergeben werden. Im Code wurde schnell klar, dass das unübersichtlich wird, sobald mehr als ein, zwei Quests dazukommen.

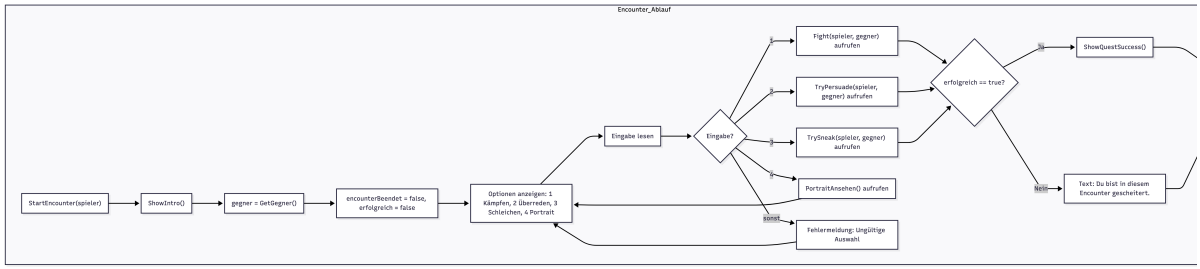
Deshalb wurden die ersten Encounters wie der Wald zunächst in eigene Dateien ausgelagert (WaldEncounter.cs), damit Intro, Kampf und Questabschluss zusammenbleiben und der switch nur noch „Wald starten“ aufruft statt die komplette Logik zu enthalten.
Aus diesem Schritt ist dann die Quests -Klasse entstanden, in der die einzelnen Quests als eigene Methoden liegen.

Die Hauptschleife ruft nur noch Quests.WaldQuest() oder Quests.RuinenQuest() auf und übergibt den Spieler. Ähnlich lief es bei der Navigation: Ursprünglich war das Menü direkt in der While-Schleife ausgeschrieben (mehrere Console.WriteLine und ReadLine direkt in Main). Um Wiederholungen zu vermeiden und Main schlanker zu machen, wurde die Menülogik in Hilfsmethoden.Questboard() ausgelagert, das nur die Auswahl anzeigt und die Eingabe zurückgibt, während die Hauptschleife die Auswahl verwertet in den Cases.

Probleme

In Program.cs war das Hauptproblem anfangs, dass viel zu viel Logik direkt in der Main hing – Quest-Aufrufe, Menüs, Textausgaben alles in einem riesigen switch. Das wurde aufgeteilt in Hilfsmethoden.Questboard() und separate Quest-Klassen, um die Main schlank zu halten. Schnell wurde der Bildschirm unübersichtlich durch lange Texte, weshalb Console.Clear() eingebaut wurde. Problem: Clear()-Aufrufe in Encounters überschrieben teilweise Quest-Anzeigen durch falsche Platzierung, und durch die “Verschachteln von Encounter,Quest und Main wurde das debugging sehr unübersichtlich.

Encounter.cs



Encounter.cs Programmablauf

Umsetzung

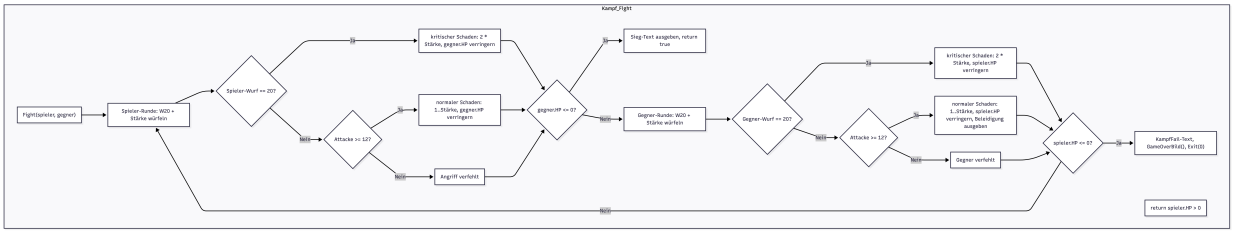
Encounter ist jetzt das Grundgerüst für jede Begegnung: Unterklassen liefern ihren Gegner über GetGegner() und füllen nur noch Intro, Überredenstext, Portrait, Game-Over-Bild und Quest-Erfolg. StartEncounter spielt die komplette Logik durch: Intro zeigen, Gegner holen, Menü anzeigen (Kämpfen/Überreden/Schleichen/Portrait), passende Methode (Fight , TryPersuade , TrySneak) aufrufen und am Ende je nach Rückgabewert Erfolgs- oder Fail-Text ausgeben. Kampf, Überreden und Schleichen laufen über Standardmethoden mit W20-Wurf + Stats und geben nur true/false zurück.

Überarbeitung

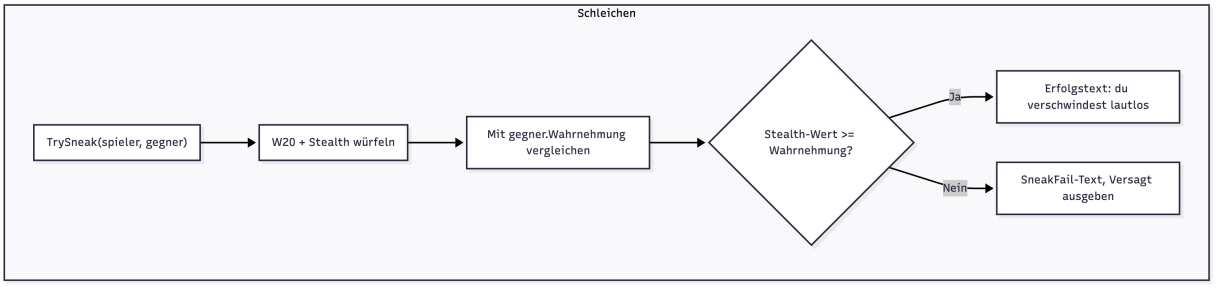
Anfangs hatte jeder Encounter seine eigene, komplett ausgeschriebene Klasse mit eigenem Kampf, eigenem Schleichen, eigener Überreden-Prüfung(Immer Copy Paste)
Dann habe Ich die Gemeinsamkeiten in eine Basisklasse gezogen, erst Start/Intro/Menu in Encounter , dann auch die Kampf-Logik zentralisiert. Später sind die Fail-Texte in Gegner gewandert, damit die Standardmethoden trotzdem pro Gegner individuelle Texte ausgeben können. Am letzten Tag habe ich die Encounter noch angepasst, dass Fails beim Schleichen & Überreden zum Kampf führen.

Probleme

- Doppelte Properties in Unterklassen (z.B. eigenes HP neben dem aus Charakter) haben dafür gesorgt, dass Gegner im Code andere Werte hatten als im Kampf. Musste bereinigt werden.
- Die Rückgabe von Erfolg/Misserfolg war zuerst überall anders gelöst; StartEncounter wusste teilweise nicht, worauf es sich verlassen kann. Das wurde vereinheitlicht auf bool-Return aus Fight / TryPersuade / TrySneak .
- Der Portrait-Case hat anfangs den Encounter beendet, weil er wie Kampf behandelt wurde – jetzt zeigt er nur das Bild und springt zurück ins Menü.



Fight Methode



TrySneak Methode

Gerüchte.cs

Umsetzung

Gerüchte hält eine statische Liste aller möglichen Gerüchte im Spiel, jeweils als kompletter Dialog-Text. Über GerüchteHören(spieler) wird ein zufälliges Ereignis angestoßen: Erst wird mit einem Wurf von 1–99 geprüft, ob der Spieler überhaupt etwas Spannendes aufschnappt – bei hohem Wurf kommt nur ein kurzer „nichts Besonderes"-Text und die Methode endet. Fällt der Wurf günstig, baut die Methode aus alleGeruechte eine Liste aller Gerüchte, die der Spieler noch nicht kennt, wählt daraus per Zufall eines aus, zeigt es farbig an und trägt es in spieler.BekannteGeruechte ein. Kennt der Spieler schon alle, gibt es eine passende Meldung und kein neues Gerücht.

Überarbeitung

Die Idee war, Gerüchte nicht hart in Encountern zu verstreuen, sondern an einer zentralen Stelle zu sammeln. Anfangs war das eher als einfache List<string> ohne Logik gedacht, später wurde daraus eine eigene Klasse mit statischer Methode, damit die Stadt, die Bar oder andere Orte einfach GerüchteHören(spieler) aufrufen können. Die Verwaltung „kenne ich schon / kenne ich noch nicht" ist dann Schritt für Schritt dazugekommen, damit der Spieler nicht ständig denselben Text liest, sondern wirklich nach und nach alle Lore-Schnipsel freischaltet.

Probleme

Ein Stolperstein war, die unbekannten Gerüchte sauber zu filtern: Wenn man direkt aus alleGeruechte zieht, ohne vorher gegen spieler. BekannteGeruechte zu prüfen, wiederholen sich Texte sehr schnell. Deshalb gibt es jetzt die Zwischensammlung unbekannteGerüchte , die jedes Gerücht nur dann aufnimmt, wenn es noch nicht in der Spieler-Liste steckt. Hier habe ich mir leider fast alles von Perplexity vorschreiben lassen, weil ich nicht auf die Idee gekommen bin zwei Gerüchtelisten anlegen zu lassen und auch dann nicht weitergekommen bin.

BesucheBar

Umsetzung

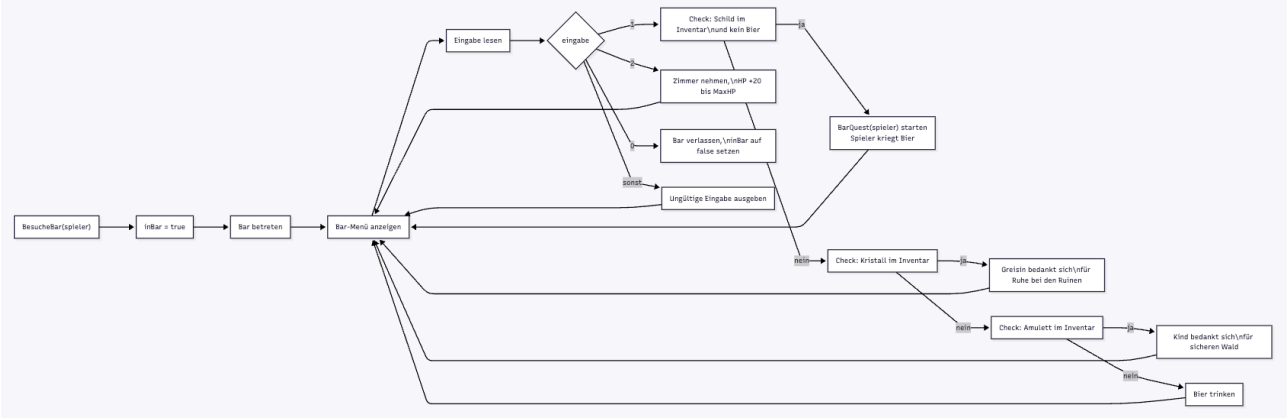
BesucheBar(spieler) öffnet ein eigenes kleines Bar-Menü mit drei Optionen: Bier bestellen, Zimmer nehmen oder die Bar verlassen. Beim Bier-Case wird zuerst geprüft, ob der Spieler das Schild, aber noch kein Bier im Inventar hat – dann startet die Bar-Quest mit Gusir Sigurdson. Hat er stattdessen den Kristall, bekommt er eine kleine Dankesszene mit der Greisin, beim Amulett spricht ihn ein Kind an. Wenn nichts davon zutrifft, trinkt der Spieler einfach schlechtes Bier und verliert HP. Die Zimmer-Option gibt etwas HP zurück (bis maximal auf das Maximum), und bei „0" wird inBar auf false gesetzt, sodass der Spieler wieder in die Stadt zurückkehrt.

Überarbeitung

Ursprünglich war die Bar nur als Ort gedacht, um Gerüchte abzuholen oder zu heilen. Mit der Zeit ist daraus ein eigener kleiner Hub mit Story-Verzweigungen geworden: Erst kam die Idee, dass der Bruder der Stadtwache (Gusir) in der Bar auf den Spieler trifft, wenn dieser das Schild trägt. Danach wurden die optionalen Flavor-Szenen eingebaut, die auf bereits gelöste Quests reagieren (Greisin für die Ruinen, Kind für den Wald), damit die Bar sich lebendig anfühlt und die Welt auf die Taten des Spielers reagiert, ohne dass dafür gleich ein eigener Encounter nötig ist.

Probleme

Die größte Falle hier war die Bedingungslogik für das Inventar: Die Reihenfolge der if / else if -Blöcke musste so gestaltet werden, dass immer der „interessanteste“ Fall zuerst greift (Schild Quest Fortsetzung, dann Kristall, dann Amulett, dann Standard-Bier). Ein falsch gesetztes else oder eine andere Reihenfolge hatte dazu geführt, dass z.B. die Bar-Quest nie startet, weil vorher schon eine andere Bedingung true wurde. Außerdem wurde die Bar Quest zuerst jedes mal gestartet, auch wenn sei schon abgeschlossen war, deshalb musste dem spieler ein Item(Bier) gegeben werden, was verhindert dass die Quest erneut ausgelöst wird. Das ganze ist keine tolle Lösung aber mir ist etwas die Zeit ausgegangen.



BesucheBar Methode

Kritik & Learnings

Rückblickend zeigt sich beim Balancing, dass die Klassenwahl den Spielstil zu stark festlegt: Wer sich einmal für Kämpfer, Barde oder Schurke entscheidet, hat später kaum interessante Entscheidungen, weil die gewählte Route meist „die richtige“ und dadurch oft zu einfach ist. Die Gerüchte liefern zwar schöne Lore, bringen spielmechanisch aber nichts; ein geplantes Feature, bei dem Gerüchte über ein Dictionary temporäre Buffs geben, konnte aus Zeitgründen nicht mehr umgesetzt werden. Dazu kommt, dass der Code deutlich mehr Kommentare bräuchte – hier wurde Dokumentation zu oft schleifen gelassen. Die Trennung in Quests und Encounter fühlt sich im Nachhinein nicht optimal an und ließe sich sicher sauberer strukturieren.

Das zentrale Learning daraus ist, sich früher und bewusster Gedanken über Struktur und Verantwortlichkeiten zu machen, statt zu viel „draufloszuprobieren“. Das Experimentieren hat zwar Spaß gemacht, aber am Ende ging viel Zeit für Aufräumen, Umbauen drauf(u.A. am letzten Tax für die Optischen Anpassungen der Konsolenausgabe). Auch die Dokumentation wurde zu spät ernst genommen: Statt während der Entwicklung mitzulaufen, musste vieles im Nachhinein rekonstruiert werden, was deutlich mühsamer und fehleranfälliger war.

Quellen

- https://www.perplexity.ai/
  - Textgenerierung für Spieltext (Prompt: kurze Skizze der Szene-> kompletter Text
  - Hilfe beim Coden (siehe Kapitel Projektablauf)
  - Dokumentation (Prompt: stichpunktartige Sätze -> Fließtext)
  - Anpassen der Mermaidcharts
- https://www.mermaidchart.com/ Für Flowcharts in der Dokumentation
- reddit.com/R/DnD Titelbild Dokumentation
- https://www.asciiart.eu/ Ingame Ascii Grafiken
- https://emojicombos.com/ Ascii Grafiken