ieses Dokument enthält detaillierte Informationen zur **physikalischen Struktur** unseres Projekts im Dateisystem und der Entwicklungsumgebung.

**1. Projektstruktur (Richtlinie)**

Wir halten uns an diese Struktur, die im Repository gepflegt wird. **Diese Struktur ist eine Richtlinie und kann jederzeit an Projektbedürfnisse angepasst, erweitert oder geändert werden.** Es ist wichtig, beim Erstellen von Code-Einheiten den korrekten Platz in dieser Struktur zu finden, um Modularität und Übersichtlichkeit zu gewährleisten.

python-rpg-projekt/

│

├── src/ # Hauptverzeichnis für den gesamten Quellcode

│ ├── config/ # Konfigurationsdateien (JSON5 + Python-Lader)

│ │ ├── \_\_init\_\_.py # Macht 'config' zu einem Python-Paket

│ │ ├── settings.json5 # Konfigurationsdatei im JSON5-Format (z.B. Spiel-Balance, Pfade)

│ │ └── config.py # Zentrales Config-Objekt lädt Einstellungen aus settings.json5

│ │

│ ├── definitions/ # Definitionen von Spielobjekten & Daten-Templates

│ │ ├── json\_data/ # Reine JSON5-Daten (Charakter-Templates, Skills, Gegner-Templates etc.) - Nutzen `//` Kommentare

│ │ │ ├── characters.json5

│ │ │ ├── opponents.json5

│ │ │ ├── skills.json5

│ │ │ ├── items.json5 # (Platzhalter)

│ │ │ ├── npcs.json5 # (Platzhalter)

│ │ │ └── quests.json5 # (Platzhalter)

│ │ ├── \_\_init\_\_.py # Macht 'definitions' zu einem Python-Paket

│ │ ├── character.py # Python-Klasse `Character` (Definition der Struktur und Basis-Verhalten eines Charakter-Templates)

│ │ ├── skill.py # Python-Klasse `Skill` (Definition der Struktur und Basis-Verhalten eines Skills)

│ │ ├── item.py # (Platzhalter) Python-Klasse `Item`

│ │ └── loader.py # Lädt JSON5-Daten (Templates) in Python-Objekte (nutzt die `json5`-Bibliothek)

│ │

│ ├── utils/ # Allgemeine Hilfsmodule und -funktionen

│ │ ├── \_\_init\_\_.py # Macht 'utils' zu einem Python-Paket

│ │ └── logging\_setup.py # Konfiguration des Logging-Systems

│ │

│ ├── game\_logic/ # Kernlogik und Spielzustandsmanagement

│ │ ├── \_\_init\_\_.py # Macht 'game\_logic' zu einem Paket

│ │ ├── formulas.py # Modul für grundlegende Berechnungsformeln (z.B. Attributbonus, HP-Berechnung)

│ │ ├── entities.py # Enthält die Klasse `CharacterInstance` (repräsentiert eine konkrete Instanz eines Charakters oder Gegners im Spiel, mit aktuellem Zustand wie HP, Statuseffekte)

│ │ ├── effects.py # Enthält die Basisklasse `StatusEffect` und konkrete Effekt-Implementierungen (mit `on\_tick`-Logik)

│ │ ├── leveling.py # Modul/Service für XP-Vergabe und Level-Up-Logik (entkoppelt von `entities.py`)

│ │ └── combat.py # Modul für die Abwicklung von Kampfaktionen, Trefferberechnung, Schadensanwendung etc.

│ │

│ ├── ai/ # Künstliche Intelligenz (Regelbasiert & RL)

│ │ ├── \_\_init\_\_.py # Macht 'ai' zu einem Paket

│ │ ├── strategies/ # Einzelne (regelbasierte) AI-Strategien

│ │ │ ├── \_\_init\_\_.py

│ │ │ ├── basic\_melee.py # Einfache Nahkampf-Strategie

│ │ │ ├── basic\_ranged.py # Einfache Fernkampf-Strategie

│ │ │ └── support\_caster.py # Einfache Unterstützer/Caster-Strategie

│ │ ├── ai\_dispatcher.py # Wählt die passende AI-Strategie basierend auf Charakterdaten

│ │ ├── rl\_training.py # (Noch nicht erstellt) Skript zum Trainieren von RL-Agenten

│ │ ├── evaluate\_agent.py# (Noch nicht erstellt) Skript zur Evaluierung von RL-Agenten

│ │ └── models/ # Verzeichnis für trainierte RL-Modelle (in .gitignore)

│ │

│ ├── environment/ # (Noch nicht erstellt) RL-Umgebung (Gymnasium-kompatibel)

│ │ ├── \_\_init\_\_.py

│ │ ├── rpg\_env.py

│ │ ├── env\_state.py

│ │ ├── observation\_manager.py

│ │ ├── action\_manager.py

│ │ └── reward\_calculator.py

│ │

│ ├── ui/ # (Noch nicht erstellt) Benutzeroberfläche (aktuell CLI-Simulation)

│ │ ├── \_\_init\_\_.py

│ │ ├── cli\_output.py

│ │ └── cli\_main\_loop.py # Steuert die automatische Simulationsschleife

│ │

│ └── main.py # HAUPTEINSTIEGSPUNKT: Parsed Argumente (--mode) und startet die Simulation/das Training

│

├── tests/ # Verzeichnis für automatisierte Tests (`pytest`)

│ └── (Noch keine Tests implementiert, Struktur sollte src/ spiegeln)

│

├── logs/ # Verzeichnis für Laufzeitprotokolle (in .gitignore, wird dyn. erstellt)

│

├── reports/ # Verzeichnis für Berichte (Analysen etc.)

│

├── docs/ # Dokumentationsverzeichnis

│ ├── ENTSCHEIDUNGEN.md # Protokoll wichtiger Entscheidungen (manuell gepflegt)

│ └── README\_DYNAMIC\_SETTINGS.md # (Noch nicht erstellt) Anleitung zur Parameteranpassung

│

├── tools/ # Hilfsskripte für Entwicklungsaufgaben

│ ├── context\_extractor.py # (Existiert evtl., nicht von uns erstellt)

│ └── (Weitere Skripte...)

│

├── .gitignore # Definiert von Git ignorierte Dateien/Ordner

└── requirements.txt # Liste der Python-Paketabhängigkeiten (\*\*muss `json5` enthalten!\*\*)

*(Statusmarkierungen wie [AKTIV]/[PLATZHALTER]/(Noch nicht erstellt) dienen nur zur Orientierung.)*

***.gitignore Beispiele (sollten in der Datei so oder ähnlich stehen):***

*venv/*

*\_\_pycache\_\_/*

*\*.pyc*

*saves/*

*logs/*

*src/ai/models/*

*\*.prof*

*.pytest\_cache/*

*reports/ # Ignoriert alle Reports, falls gewünscht*

***2. GitHub/Codespaces Workflow & Umgebung***

* ***Versionskontrolle:*** *Nutze Git konsequent (add, commit, push, pull). Feature-Branches und Pull Requests sind empfohlen. Repo:* [*https://github.com/Kharas81/python-rpg-projekt*](https://github.com/Kharas81/python-rpg-projekt)
* ***Entwicklungsumgebung:*** *Primär GitHub Codespaces oder VS Code (mit GitHub).*
* ***Virtuelle Umgebung:*** *Codespaces stellt oft eine Umgebung bereit. Stelle sicher, dass*  ***(die jetzt***  *im Terminal ausgeführt wird.*
* ***.gitignore:*** *Wichtig! Stelle sicher, dass sensible oder temporäre Dateien/Ordner ignoriert werden (siehe Beispiele oben).*
* ***Dateierstellung/Code-Lieferung:******Gemäß Priorität 4 im Master-Core:***
  1. *Dateien/Verzeichnisse per cat EOF oder mkdir/touch erstellen.*
  2. *Vollständigen Code-Inhalt danach als separaten Code-Block liefern.*
* ***Ausführung:*** *Im Terminal (im Projekt-Root), z.B. python src/main.py --mode manual oder python src/main.py --mode auto*