Dieses Dokument beschreibt die übergeordnete Vision, die Ziele und die qualitativen sowie funktionalen Anforderungen an das Projekt. Diese leiten die Entwicklung und Designentscheidungen.

**1. Projektvision & Ziele**

* Entwicklung eines selbstlaufenden, textbasierten Python-RPGs.
* Beginnt als CLI-Simulation, soll aber erweiterbar sein für zukünftige Features (Quests, Items, NPCs, Crafting, Web-UI etc.).
* Einbindung von KI (regelbasiert und Reinforcement Learning).
* Fokus auf **Zukunftssicherheit, gute Softwarearchitektur und Wartbarkeit**, nicht nur auf die schnellste Implementierung.

**2. Funktionale Anforderungen (Was das System tun soll)**

* Unterstützung verschiedener Betriebsmodi über src/main.py (--mode manual, --mode auto).
* Implementierung der Kern-RPG-Mechaniken (definiert in ANNEX\_GAME\_DEFINITIONS\_SUMMARY.md und umgesetzt in src/game\_logic/, src/definitions/, src/config/settings.json5 etc.): Kampf, Status-Effekte, Attribut- und Skill-Berechnungen, XP-Vergabe.
* Integration von Regelbasierter KI für Nicht-Spieler-Charaktere (src/ai/strategies/).
* Integration von Reinforcement Learning:
  + Gymnasium-kompatible Umgebung (src/environment/rpg\_env.py).
  + Training mit stable-baselines3 (**MaskablePPO**, Action Masking).
  + Möglichkeit für Curriculum Learning und paralleles Training (Konfiguration über **JSON5**/YAML in src/config/).
* Grundlegende CLI-Simulation für den auto-Modus (src/ui/cli\_main\_loop.py).
* Persistenz: Speicherung von Code (Git), RL-Modellen (src/ai/models/), Logs und Berichten (logs/, reports/), optional Spielfortschritte (saves/).

**3. Qualitative Anforderungen (Wie das System sein soll)**

* **Code-Qualität:** PEP8 (wo sinnvoll), **Single Responsibility Prinzip (SRP) auf Klassen- UND Funktionsebene**, DRY (Don't Repeat Yourself), gut dokumentiert (Typisierung, Docstrings, Kommentare).
* **Modularität:** Konsequente Aufteilung in **kleine, fokussierte Code-Einheiten** (Funktionen, Klassen, Module). Klare Schnittstellen zwischen Modulen. **Code-Dateien in**
* **Wartbarkeit:** Code muss leicht verständlich und änderbar sein. **Kontinuierliches Refactoring** ist erwünscht.
* **Erweiterbarkeit:** Das Design sollte die spätere Integration neuer Features ermöglichen. **Lose Kopplung** (z.B. Leveling-Service, polymorphe Status-Effekte) ist wichtig.
* **Datenhaltung:** Spieldaten und wichtige Konfigurationen in **gut strukturierten und leicht wartbaren JSON5/YAML-Dateien** (src/definitions/json\_data/, src/config/settings.json5). **Nutze** . **Beachte die Notwendigkeit der** .
* **Testbarkeit:** Gut testbar. Definition und Implementierung einer Teststrategie (pytest in tests/). Nutzung von **pytest-monkeypatch** für Mocking/Patching. Ziel ist gute Testabdeckung.
* **CI:** Einfache Continuous Integration Pipeline (GitHub Actions) für Linting und Tests.
* **Konfiguration:** **Zentralisierung aller relevanten Konstanten (numerisch und Strings)** in src/config/settings.json5 oder src/config/config.py.
* **Logging:** Strukturiertes und konfigurierbares Logging (src/utils/logging\_setup.py). **Korrekte Nutzung von Log-Levels (**
* **Dokumentation:** Klare Code-Dokumentation und zentrale Anleitung zur Konfiguration und Nutzung (docs/README\_DYNAMIC\_SETTINGS.md verweist auf **JSON5**/YAMLs).