■ Projektablaufplan – Monsterkampf-Simulator (K1, S1, S4)

Phase 1: Planung & Setup

Ziel: Technische Basis schaffen und Struktur vorbereiten

- Projektstrukturplan erstellen
- Entwicklungsumgebung (Visual Studio, .NET 8.0) einrichten
- Konsolenprojekt anlegen
- Namensräume, Ordner und Grundstruktur definieren
- Testausgabe („Hello World“) prüfen

Ergebnis:

→ Lauffähiges Grundgerüst mit klarer Struktur

Phase 2: Kernarchitektur (CoreManager + Manager-Schicht)

Ziel: Fundament für den Programmfluss erstellen

- `CoreManager` als zentralen Einstiegspunkt implementieren
- `BattleManager`, `RulesManager`, `InputManager` anlegen (Grundgerüste, noch ohne Logik)
- `DiagnosticsManager` und `RandomManager` vorbereiten
- Erste Testverbindungen zwischen Managern prüfen

Ergebnis:

→ Programm startet und ruft Manager in richtiger Reihenfolge auf

---

Phase 3: Monster-System (OOP-Kern)

Ziel: Monsterstruktur und Vererbung aufbauen

- `MonsterBase` (abstract) mit HP, AP, DP, S, Race erstellen
- Abgeleitete Klassen (`Orc`, `Troll`, `Goblin`) anlegen
- `MonsterMeta`-Datenstruktur (Stats) einführen
- `TakeDamage()` und `Attack()` implementieren
- Erste Tests mit zwei Monstern durchführen

Ergebnis:

→ Zwei Monster können erzeugt werden und Schaden verursachen

Phase 4: Skill-System

Ziel: Angriff- und Fähigkeitssystem implementieren

- `SkillBase` (abstract) erstellen
- Unterklassen: aktive und passive Skills

- Beispiel: `FireSpell`, `WaterSpell`, `PassiveSkill_Rage`
- Skills den Monstern zuweisen
- Damage-Formel und kritische Treffer testen

Ergebnis:

→ Monster besitzen individuelle Skills / Attacken

Phase 5: Kampfmechanik

Ziel: Kampfrunden und Siegbedingungen umsetzen

- `BattleManager.StartBattle()` implementieren
- Rundenlogik programmieren (abwechselnde Attacken)
- Initiative anhand Geschwindigkeit (S) bestimmen
- Abbruch bei HP ≤ 0
- `UIManager`: Kampfverlauf und Ergebnis ausgeben

Ergebnis:

→ Vollständig spielbarer Konsolenkampf

Phase 6: Validierung & Regeln

Ziel: Stabilität und Eingabelogik absichern

- `RulesManager`: Kampfregeln prüfen (z. B. keine gleiche Rasse)
- `InputManager`: Benutzereingaben validieren
- Endlosschleifen vermeiden
- `RandomManager`: Seed und zufällige Fähigkeiten

Ergebnis:

→ Fehlerfreier, kontrollierter Spielablauf

Phase 7: Debugging & Refactoring

Ziel: Codequalität und Lesbarkeit erhöhen

- `DiagnosticsManager`: Fehler, Warnungen, Checks
- Methoden kommentieren (XML)
- Redundanzen entfernen (SRP-Prüfung)
- Klassen strukturieren und Aufteilung optimieren
- Funktionstests mit verschiedenen Kombinationen

Ergebnis:

→ Saubere, wartbare Codebasis

Phase 8: Dokumentation & Abgabe

Ziel: Projektabschluss und Nachvollziehbarkeit

- README mit Spielbeschreibung und Bedienung
- PSP, Ablaufplan und Zeitplan ergänzen
- Feedback einholen und bewerten
- Projekt auf GitHub hochladen / SAE abgeben

Ergebnis:

→ Abgabereifes, vollständig dokumentiertes Projekt