Projektbericht: Unity Labyrinthspiel

Joël Deffner

6. Juli 2025

1 Einleitung

Die Aufgabe bestand darin, einen nicht spielbaren Charakter (NPC) für ein bestehendes Labyrinthspiel zu entwickeln, dessen Verhalten durch eine integrierte Sprach-KI situationsabhängig gesteuert wird. Der NPC sollte menschenähnliches Verhalten zeigen, verschiedene Bewegungsarten beherrschen und als ängstlicher Reisebegleiter des Spielers fungieren. Das Hauptziel war die Entwicklung eines KI-gesteuerten NPCs, der als ängstlicher Reisebegleiter dem Spieler durch ein dynamisch generiertes Labyrinth folgt. Der NPC sollte situationsabhängig reagieren, verschiedene Bewegungsgeschwindigkeiten aufweisen und durch eine lokale Sprachmodell-Integration natürliches Verhalten zeigen. Dabei würde das Ziel gesetzt mindestens drei Bewegungsarten implementiert werden: langsames Gehen, normales Folgen und schnelles Weglaufen bei Angst. Diese Dokumentation beschreibt den Entwicklungsprozess, die implementierte Funktionalität sowie die technischen Herausforderungen, insbesondere bei der Integration der KI-Steuerung.

2 Projektübersicht

2.1 Technische Grundlage

Das Projekt basiert auf Unity 3D mit C# als Programmiersprache. Die KI-Integration erfolgt über das Neocortex Ollama-Framework, welches eine lokale Kommunikation mit Sprachmodellen ermöglicht. Zur Entwicklungsunterstützung wurde GitHub Copilot eingesetzt, was insbesondere bei der Implementierung der KI-Logik und der iterativen Verbesserung des Codes hilfreich war.

3 Entwicklungsprozess und Implementierung

3.1 Iterative Entwicklungsmethodik

Der Entwicklungsprozess folgte einem iterativen Ansatz mit mehreren Überarbeitungszyklen. Durch die Verwendung von GitHub Copilot konnte die Entwicklungsgeschwindigkeit erhöht und die Code-Qualität verbessert werden, insbesondere bei der Implementierung wiederkehrender Programmiermuster und der KI-Integration.

3.2 Erste Implementierungsphase: Grundlegende Bewegungslogik

Zunächst wurde ein einfaches Bewegungssystem entwickelt, das grundlegende Bewegungs- und Verfolgungsmechanismen umfasste. Diese erste Version (`SimpleAI.cs`) implementierte eine traditionelle Zustandsmaschine mit fest programmierten Verhaltensweisen. Das System verwendete Raycast-basierte Hindernisvermeidung und dynamische Geschwindigkeitsanpassung basierend auf der Entfernung zum Spieler.

3.3 KI-Integration: Die zentrale Herausforderung

Die Integration der Sprachmodell-Steuerung stellte die größte technische Herausforderung des Projekts dar. Die Komplexität lag nicht nur in der technischen Implementierung der API-Kommunikation, sondern auch in der Entwicklung eines strukturierten Kommunikationsprotokolls zwischen dem Unity-System und einem Sprachmodell.

Das entwickelte System verwendet ein standardisiertes Antwortformat: `geschwindigkeit,richtung,nachricht`. Hierbei kontrolliert die KI direkt die Bewegungsgeschwindigkeit (0-7), die Bewegungsrichtung (zum Spieler hin, vom Spieler weg, oder Patrouille) sowie die emotionale Reaktion des NPCs. Diese direkte Kontrolle

ermöglicht es der KI, authentische Verhaltensweisen zu generieren, die über einfache Skriptlogik hinausgehen.

3.4 Finale Implementierung: KI-gesteuerter Reisebegleiter

Die endgültige Lösung implementiert einen ängstlichen Reisebegleiter, dessen Verhalten vollständig durch ein lokales Sprachmodell gesteuert wird. Der NPC analysiert kontinuierlich seine Situation relativ zum Spieler und sendet kontextuelle Informationen an die KI.

Listing 1: KI-gesteuerte Bewegungslogik

```
void HandleAIMovement()
{
    Vector3 targetPosition;

    if (moveTowardsPlayer && player != null)
    {
        // Folge dem Spieler in angemessener Entfernung
        Vector3 directionToPlayer = (player.position - transform.position).normalized;
        float distanceToPlayer = Vector3.Distance(transform.position, player.position);

        if (distanceToPlayer > 4f)
            targetPosition = player.position;
        else if (distanceToPlayer < 2f)
            targetPosition = transform.position - directionToPlayer * 0.5f;
        else
            targetPosition = player.position;
    }
    else if (moveAwayFromPlayer && player != null)
    {
        // Laufe weg wenn veraengstigt
        Vector3 awayDirection = (transform.position - player.position).normalized;
        targetPosition = transform.position + awayDirection * 8f;
    }

    if (!isStopped && currentSpeed > 0f)
        MoveTowards(targetPosition);
}
```

3.5 KI-Kommunikationsprotokoll

Das Herzstück der Implementierung ist das Kommunikationsprotokoll zwischen Unity und dem Sprachmodell. Alle paar Sekunden wird der aktuelle Zustand des NPCs analysiert und als kontextueller Prompt an die KI gesendet:

Listing 2: Situationsanalyse und KI-Kommunikation

```
void SendSituationToAI()
{
    float distanceToPlayer = Vector3.Distance(transform.position, player.position);
    string situation = "";

    if (distanceToPlayer <= 2f)
        situation = "Ich bin direkt neben meinem Begleiter. Ich fuehle mich sicherer...";
    else if (distanceToPlayer <= 5f)
        situation = "Ich folge meinem Begleiter in gutem Abstand...";
    else if (distanceToPlayer > 15f)
        situation = "Ich kann meinen Begleiter kaum noch sehen! Ich habe Angst...";

    request.Send(situation);
}
```

4 Technische Herausforderungen und Schwierigkeiten

4.1 KI-Integration als Hauptherausforderung

Die Integration der Sprachmodell-Steuerung stellte zweifellos die größte technische Herausforderung des gesamten Projekts dar. Die Schwierigkeit lag nicht nur in der reinen API-Implementierung, sondern

vielmehr in der Entwicklung eines robusten Kommunikationssystems zwischen dem deterministischen Unity-Environment und dem probabilistischen Verhalten der KI.

Die erste große Hürde bestand in der Etablierung eines zuverlässigen Datenformats für die KI-Antworten. Sprachmodelle tendieren dazu, kreative und unstrukturierte Antworten zu generieren, was für ein Spielsystem problematisch ist, das klare, parsbare Befehle benötigt. Die Lösung lag in der Entwicklung eines strikten Antwortprotokolls mit Fallback-Mechanismen für fehlerhafte oder unerwartete Antworten.

Ein weiteres bedeutendes Problem war die Latenz der KI-Kommunikation. Während der Entwicklung zeigte sich, dass häufige API-Aufrufe die Spielerfahrung erheblich beeinträchtigen können. Die implementierte Lösung verwendet ein Intervall-basiertes System, das alle fünf Sekunden den Zustand analysiert und nur bei signifikanten Änderungen neue Anfragen sendet.

4.2 Prompt Engineering und Verhaltenskonsistenz

Eine unerwartete Komplexität lag im sogenannten "Prompt Engineering der Gestaltung der Systemnachrichten, die das Verhalten der KI definieren. Es stellte sich heraus, dass die Persönlichkeit und das Verhalten des NPCs stark von der Formulierung dieser initialen Anweisungen abhängen. Mehrere Iterationen waren nötig, um ein konsistentes Verhalten des ängstlichen Reisebegleiters zu erreichen.

4.3 Performance und Ressourcenmanagement

Die Integration lokaler Sprachmodelle über das Ollama-Framework brachte zusätzliche Performance-Überlegungen mit sich. Während die lokale Ausführung Vorteile bezüglich Datenschutz und Latenz bietet, musste das System so gestaltet werden, dass es auch bei temporären KI-Ausfällen funktionsfähig bleibt. Das Gemma 3 4B-Modell wurde gewählt, da es eine gute Balance zwischen Performance und Qualität bietet.

4.4 Navigation und Bewegungssteuerung

Die Implementierung einer effizienten Navigation in dynamisch generierten Labyrinthen erwies sich als komplexer als initially erwartet. Während Unity's NavMesh-System für statische Umgebungen optimal ist, zeigten sich bei dynamisch generierten Strukturen erhebliche Limitationen. Die Notwendigkeit, Navigationsgitter zur Laufzeit zu berechnen, führte zu Performance-Problemen und unvorhersagbaren Verhaltensweisen.

Die entwickelte Alternative verwendet ein Raycast-basiertes System für die Hindernisvermeidung. Diese Lösung ist zwar weniger sophisticated als ein vollständiges Pathfinding-System, erweist sich jedoch als robust und performant für die Anforderungen des Projekts:

Listing 3: Raycast-basierte Hindernisvermeidung

```
if (!Physics.Raycast(transform.position + Vector3.up * 0.5f, direction, 1f))
{
    // Freier Weg - normal bewegen
    transform.position += direction * currentSpeed * Time.deltaTime;
}
else
{
    // Hindernis erkannt - alternative Routen testen
    Vector3 rightDirection = Quaternion.Euler(0, 45, 0) * direction;
    Vector3 leftDirection = Quaternion.Euler(0, -45, 0) * direction;

    if (!Physics.Raycast(transform.position + Vector3.up * 0.5f, rightDirection, 1f))
        transform.position += rightDirection * currentSpeed * 0.5f * Time.deltaTime;
    else if (!Physics.Raycast(transform.position + Vector3.up * 0.5f, leftDirection, 1f))
        transform.position += leftDirection * currentSpeed * 0.5f * Time.deltaTime;
}
```

5 Projektreflexion und Erkenntnisse

5.1 Entwicklungsmethodik und Tooling

Die Verwendung von GitHub Copilot erwies sich als besonders wertvoll bei der iterativen Entwicklung und der Implementierung wiederkehrender Programmiermuster. Insbesondere bei der KI-Integration konnten

durch die KI-Assistenz komplexe API-Integrationen und Fehlerbehandlungsroutinen effizienter entwickelt werden. Dies demonstriert die Bedeutung moderner Entwicklungstools für die Bewältigung komplexer technischer Herausforderungen.

5.2 KI-Integration in Echtzeitsystemen

Die Arbeit an diesem Projekt verdeutlichte sowohl das Potenzial als auch die aktuellen Limitationen von Sprachmodellen in interaktiven Anwendungen. Während die KI durchaus in der Lage ist, konsistente und nachvollziehbare Verhaltensweisen zu generieren, erfordert die Integration erhebliche Aufmerksamkeit für Fehlerbehandlung, Performance und Benutzererfahrung.

Die entwickelte Lösung zeigt, dass KI-gesteuerte NPCs bereits heute realisierbar sind, jedoch eine sorgfältige Balance zwischen KI-Autonomie und systemischer Kontrolle erfordern. Zukünftige Entwicklungen in der Sprachmodell-Technologie werden voraussichtlich sowohl die Latenz als auch die Konsistenz solcher Systeme verbessern.

5.3 Praktische Implementierungserkenntnisse

Ein wichtiger Lerneffekt war die Erkenntnis, dass die technische Implementierung der KI-Integration nur einen Teil der Herausforderung darstellt. Mindestens ebenso wichtig ist das Design der Interaktionsparadigmen und die Definition klarer Verhaltensrichtlinien für die KI. Das entwickelte Protokoll mit seinem strukturierten Antwortformat erwies sich als essential für die Systemstabilität.

6 Fazit und Ausblick

Das Projekt demonstriert erfolgreich die Machbarkeit KI-gesteuerter NPCs in Unity-basierten Spielumgebungen. Trotz der erheblichen technischen Herausforderungen, insbesondere bei der KI-Integration, konnte ein funktionsfähiges System entwickelt werden, das einen authentisch wirkenden, ängstlichen Reisebegleiter implementiert.

Die entwickelte Lösung erfüllt die ursprünglichen Anforderungen der Aufgabenstellung: Der NPC zeigt situationsabhängiges Verhalten, beherrscht verschiedene Bewegungsarten (langsames Gehen, normales Folgen, schnelles Weglaufen) und reagiert menschenähnlich auf verschiedene Spielsituationen. Die Integration des lokalen Sprachmodells über das Ollama-Framework ermöglicht es dem NPC, authentische emotionale Reaktionen zu zeigen, die über einfache Skriptlogik hinausgehen.

Für zukünftige Projekte bietet das entwickelte Framework eine solide Grundlage für erweiterte KI-gesteuerte Verhaltensweisen. Mögliche Erweiterungen könnten komplexere Emotionssysteme, Gruppenverhalten für mehrere NPCs oder adaptive Lernmechanismen umfassen. Die modulare Architektur des Systems ermöglicht es, solche Features ohne grundlegende Umstrukturierung zu integrieren.

Die Erfahrungen dieses Projekts unterstreichen die Bedeutung einer durchdachten Systemarchitektur und robusten Fehlerbehandlung bei der Integration von KI-Technologien in interaktive Anwendungen. Während die vollständige Autonomie von KI-gesteuerten Charakteren noch Zukunftsmusik ist, zeigt dieses Projekt, dass bereits heute überzeugende und engaging NPCs mit Hilfe moderner Sprachmodelle realisiert werden können.