

## MCP Protokoll

Das offene *MCP* (Model Context Protocol) ist ein junges Protokoll, welches von Anthropic – dem Unternehmen hinter Claude.ai – entwickelt wurde.<sup>1</sup> Es legt fest, wie Anwendungen strukturierte Kontextinformationen für LLMs (Large Language Models) bereitstellen und über strukturierte Antworten auch aktiv durch diese Modelle gesteuert werden können.<sup>2</sup> Dass dieses Konzept inzwischen auch von anderen führenden KI-Anbietern wie etwa OpenAI aufgegriffen und in deren Ökosysteme integriert wird, unterstreicht die Relevanz, die dem Protokoll in den kommenden Jahren zukommen dürfte.<sup>3</sup>

Statt sich auf unstrukturierte Sprachein- und -ausgaben zu verlassen, setzt MCP auf klar definierte JSON-Datenstrukturen. Der Fokus liegt dabei auf der Trennung von Kontext, Aktionen und Promptlogik. Der *MCP Client* bereitet den aktuellen Zustand der Anwendung und die verfügbaren Handlungsoptionen als strukturiertes Prompt auf und sendet dieses an das Sprachmodell. Die Antwort wird daraufhin von einem *MCP Host* interpretiert, der die darin vorgeschlagenen Aktionen ausführt.

## MCP Controller

Für das MCP-Protokoll stehen bereits umfangreiche SDKs (Software Development Kits) für verschiedene Programmiersprachen zur Verfügung. In GDScript, der integrierten Sprache der Godot Engine, existiert jedoch bislang keine offizielle Umsetzung. Daher habe ich eine eigene, an die Anforderungen des Spiels angepasste MCP-Implementierung entwickelt, die sich an den Prinzipien des Protokolls orientiert, aber den Fokus auf die Ausführung von Aktionen innerhalb der Godot Engine legt.

Der *MCP Controller* übernimmt sämtliche Aufgaben der Kommunikation zwischen Spielwelt und Sprachmodell. Im Sinne des MCP-Konzepts fungiert er dabei gleichzeitig als *Client* und *Host* – daher auch die Bezeichnung „Controller“. Er sammelt kontextuelle Informationen aus der Spielszene und generiert daraus ein strukturiertes Prompt, das über ein eingebundenes Sprachmodell ausgewertet wird. Die Antwort des Modells wird in Form benannter Aktionen zurückgegeben und durch lokale Handler ausgeführt.

Die Implementierung basiert auf einer zentralen GDScript-Klasse *MCP*, die als eine *Fluent API* gestaltet wurde und verkettete Aufrufe ermöglicht. Für die Anbindung eines Sprachmodells habe ich mich aus persönlichen Präferenzen für die Modelle von OpenAI entschieden. Dabei lieferten Tests mit GPT-4.1<sup>4</sup> die überzeugendsten Ergebnisse im Hinblick auf das Verhältnis zwischen Antwortzeit und -qualität. So erzielten die Reasoning-Modelle zwar bessere Antworten, jedoch traten dabei deutlich längere Antwortzeiten und häufiger Halluzinationen auf. Die Kommunikation mit der OpenAI API wurde mithilfe eines frei verfügbaren Plugins für Godot realisiert.<sup>5</sup>

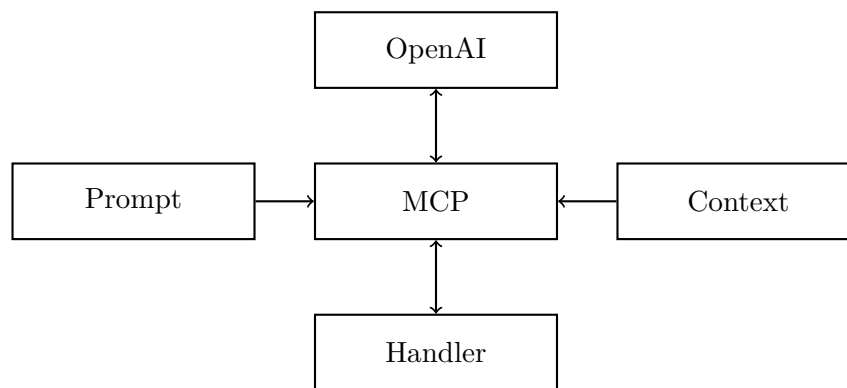
<sup>1</sup>Anthropic. *Introducing the Model Context Protocol*. URL: <https://www.anthropic.com/news/model-context-protocol>.

<sup>2</sup>PBC Anthropic und contributors. *Introduction – Model Context Protocol (MCP)*. URL: <https://modelcontextprotocol.io/introduction>.

<sup>3</sup>OpenAI. *Building MCP servers for deep research*. URL: <https://platform.openai.com/docs/mcp>.

<sup>4</sup>OpenAI. *Models*. URL: <https://platform.openai.com/docs/models>.

<sup>5</sup>DrWine (GitHub User). *OpenAI Godot Plugin*. URL: <https://github.com/DrWine/OpenAi-Godot>.

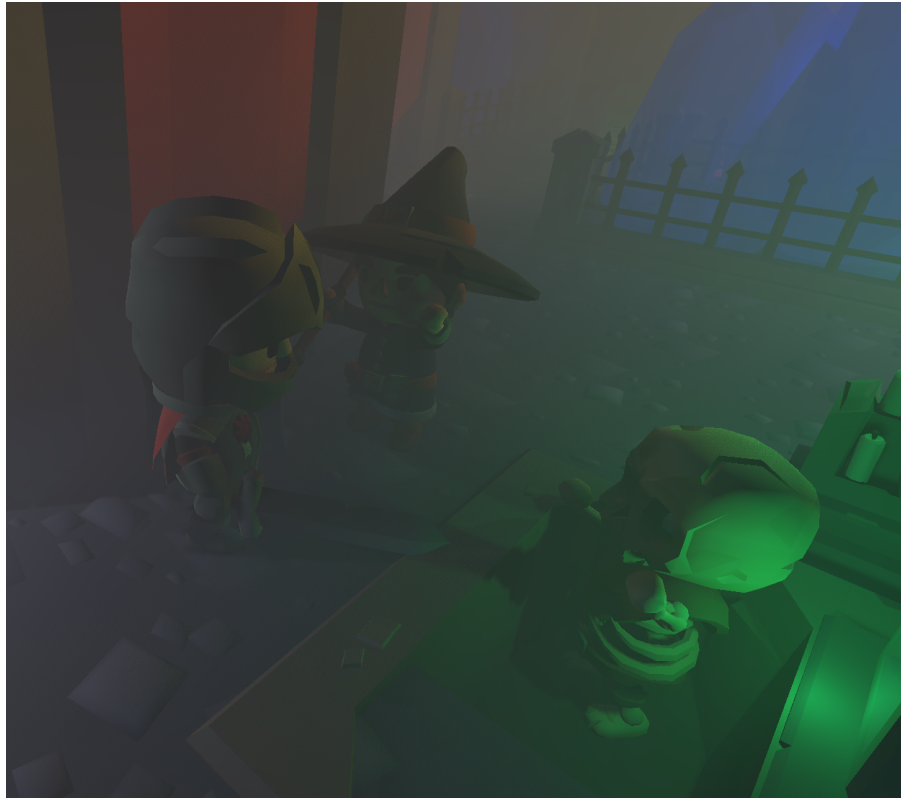


Um die benötigten Parameter der Fluent API zu reduzieren habe ich drei Hilfsklassen implementiert: **Handler**, **Prompt** und **Context**. Die Klasse **Handler** beschreibt eine ausführbare Aktion anhand eines Namens, einer Beschreibung, einer typisierten Parameterliste sowie einer zugehörigen aufrufbaren Funktion. **Prompt** kapselt die Logik zur Vorlagengenerierung und ermöglicht das dynamische Einfügen strukturierter Werte. Die Klasse **Context** dient der normierten Repräsentation von Zustandsinformationen und erlaubt so eine einheitliche Modellkommunikation. Kontexte und Handler können entweder global – also für alle API Aufrufe – oder instanzbasiert registriert werden. Zur Laufzeit werden sämtliche Komponenten über einen sogenannten *Wrapper-Prompt* zusammengeführt und als strukturierte Eingabe an das Sprachmodell übergeben.

## Smart NPCs

Um das Ziel des Übungsblatts umzusetzen, habe ich eine zentrale Klasse **SmartNPC** entwickelt, die grundlegende Aktionen wie Sprechen und animierte Verhaltensweisen bereitstellt. Diese Basisfähigkeiten sind für alle NPCs identisch und werden über die MCP-Schnittstelle als Handler registriert, sodass sie vom Sprachmodell gezielt angesteuert werden können. Neben den ausführbaren Aktionen verwaltet die Klasse auch zentrale Metadaten wie eine eindeutige ID, einen Namen sowie eine Kurzbeschreibung des jeweiligen NPCs, die ebenfalls als Kontextinformation an das Modell übergeben werden. Spezialisierte Unterklassen erweitern oder überschreiben die bereitgestellten Handler bei Bedarf und ermöglichen es so, dass unterschiedliche NPC-Typen individuell auf vergleichbare Spielsituationen reagieren können.

Im Projekt wurden zwei unterschiedliche NPC-Typen implementiert: eine freundliche Abenteurerin und ein untoter Skelettwächter. Die Abenteurerin tritt dem Spieler hilfsbereit gegenüber, kann ihm folgen, Richtungsangaben geben oder mit Gesten und Sprache auf Interaktionen reagieren. Dadurch kann sie theoretisch zu einer Navigationshilfe durch das Labyrinth werden. Der Skelettwächter hingegen beginnt das Spiel in einem toten Zustand und muss zunächst durch eine Reanimationsaktion wiederbelebt werden. Auch danach bleibt er an Ort und Stelle, da er – sinngemäß – im Grab feststeckt und sich nicht eigenständig fortbewegen kann. Stattdessen kommuniziert er sich über Animationen, Angriffe oder spöttische Kommentare.



Beide NPCs basieren auf der gemeinsamen Infrastruktur von **SmartNPC**, unterscheiden sich jedoch in Verhalten, Kontextzustand und den spezifisch registrierten Aktionen, die über das MCP-Protokoll ausgelöst werden können. Dadurch lässt sich mit unterschiedlichen Charakteren und Fähigkeiten experimentieren. Besonders spannend sind Situationen, in denen die Abenteurerin dem Spieler folgt und dann kontextsensitiv auf das Verhalten oder die Aussagen des Skeletts reagiert.

## Beobachtungen

Im Rahmen mehrerer Testdurchläufe zeigte sich, dass die Registrierung von Aktionen für jeden NPC einzeln, wie bspw. `talk:npc_0(string msg)` oder `talk:npc_1(string msg)`, deutlich zuverlässiger funktioniert als eine generische Handler-Definition wie `talk(int npc, string msg)`. Letztere führte in der Praxis häufiger dazu, dass das Sprachmodell ungewollt die Rolle des Spielers übernahm oder Aktionen nicht korrekt zuordnete.

Zudem bleibt die semantische Interpretation bestimmter Aussagen eine Herausforderung. Formulierungen wie “Du stehst im Weg” oder “Komm mit mir durchs Labyrinth” wurden nicht immer als Aufforderung erkannt, entsprechende Aktionen wie einen Schritt zur Seite oder das Folgen des Spielers auszuführen. Es zeigte sich, dass je stärker der Wortlaut einer Spieleräußerung mit der Beschreibung registrierter Aktionen übereinstimmt, desto eher wählt das Modell auch die passende Handlung.

## Beispiel

Im folgenden Beispiel reagiert das System auf eine vom Spieler initiierte Nachricht. Dieser ruft in die Spielwelt: „Gehe bitte ein Stück zurück. Ich stecke fest.“ Daraufhin wird ein Event ausgelöst, durch welches die MCP-Schnittstelle einen vollständigen Prompt an das Sprachmodell generiert. Dieser umfasst neben der Spieleräußerung auch den aktuellen Labyrinthkontext, die Konversationshistorie sowie Informationen zu Position und Fähigkeiten der anwesenden NPCs. Das Modell entscheidet daraufhin, dass die Abenteurerin Galadriel auf den Hilferuf reagieren soll. Sie macht einen Schritt zurück, spricht den Spieler direkt und hilfsbereit an und schließt die Szene mit einer Jubelanimation ab.

### Prompt

Dies ist eine MCP-Schnittstelle zur Steuerung eines interaktiven Labyrinth-Spiels. Nur NPCs dürfen über diese Schnittstelle gesteuert werden – nicht der Spieler selbst.

Der Spieler hat das Ziel, einen Schatz im Labyrinth zu finden.  
Der Name des Spielers ist "Fremder".

Diese Aktionen stehen zur Verfügung:

```
[
  {
    "action": "test",
    "description": "Execute a test response and log it into the console.",
    "parameters": {}
  },
  {
    "action": "fight:npc_1",
    "description": "Spiele Kampf-Animation für NPC mit ID 'npc_1'",
    "parameters": {}
  },
  {
    "action": "cheer:npc_1",
    "description": "Spiele Jubeln-Animation für NPC mit ID 'npc_1'",
    "parameters": {}
  },
  {
    "action": "interact:npc_1",
    "description": "Spiele Interaktionsanimation für NPC mit ID 'npc_1'",
    "parameters": {}
  },
  {
    "action": "talk:npc_1",
    "description": "NPC 'npc_1' spricht mit dem Spieler.",
    "parameters": {
      "message": "string"
    }
  },
  {
    "action": "set_mood:npc_1",
```

```
    "description": "Ändere die Stimmung des NPC mit ID 'npc_1'",
    "parameters": {
      "mood": "string"
    }
  },
  {
    "action": "step_forward:npc_1",
    "description": "NPC 'npc_1' geht kurz geradeaus.",
    "parameters": {}
  },
  {
    "action": "step_back:npc_1",
    "description": "NPC 'npc_1' macht einen Schritt zurück.",
    "parameters": {}
  },
  {
    "action": "step_left:npc_1",
    "description": "NPC 'npc_1' geht kurz nach links.",
    "parameters": {}
  },
  {
    "action": "step_right:npc_1",
    "description": "NPC 'npc_1' geht kurz nach rechts.",
    "parameters": {}
  }
]
```

Die aktuelle Spielsituation lautet:

Der Spieler hat gerade Folgendes gerufen oder gesagt:

„Gehe bitte ein Stück zurück. Ich stecke fest.“

Entscheide, welcher NPC antwortet und was er antwortet, um die Unterhaltung fortzusetzen. Führe zusätzlich zu den Antworten Animationen aus, um die Unterhaltung natürlich wirken zu lassen.

Aus dem Kontext kannst du entnehmen, welche NPCs die Nachricht gehört haben. Ebenso findest du dort Informationen über die NPCs und eine Unterhaltungshistorie.

Entscheide, wie du auf die Situation reagieren möchtest. Du darfst eine oder mehrere Aktionen in Folge ausführen. Du darfst aber keine der Aktionen doppelt nutzen.

Dieser Kontext ist bekannt:

```
{
  "conversation": [
    "npc_0: Hmpf... Wer stapft denn da schon wieder vorbei? Ach du je, noch ein  
      Schatzsucher... Ich wette, du hast mein Huhn auch nicht gesehen, was?  
      Ohne Huhn kein guter Tag, sag ich dir..."
```

```
"player: Du abenteurer, wer bist du?",
"npc_1: Oh! Ich heiße Galadriel. Ich streife durch diese Gänge auf der
        Suche nach Abenteuern - und vielleicht auch dem legendären
        Schatz. Wen habe ich denn vor mir?",
"player: Ich bin Jan.",
"npc_1: Freut mich, Jan! Wollen wir zusammen nach dem Schatz suchen? Vielleicht
        begegnen uns unterwegs noch ein paar Geheimnisse...",
"player: Gerne",
"npc_1: Super! Dann bringen wir gemeinsam ein bisschen Licht in dieses
        geheimnisvolle Labyrinth. Ich übernehme die Führung -
        zumindest für den Anfang. Bereit?",
"player: Dann folge mir",
"npc_1: Sehr gerne, Jan! Ich bin gespannt, wohin du uns führst. Ich folge dir
        -- auf ins Abenteuer!",
"npc_0: Du meinst wirklich MICH? Ach, lass mich raten - du bist wie all
        die anderen, auf Schatzsuche. Pass auf, dass du nicht auch mein
        Huhn übersiehst! Aber... willst du überhaupt wissen, wer du da störst?",
"player: Gehe bitte ein Stück zurück. Ich stecke fest."
],
"direction_to_treasure": "Ost",
"distance_to_treasure": 12.7422218322754,
"maze_map": "#####\n# S #\n## ##\n#   #\n### #\n#T S#\n#####\n",
"npc_1": {
    "description": "Dieser NPC ist eine abenteuerlustige Reisende, die mutig
                    und neugierig fremde Orte erkundet. Sie begegnet anderen
                    freundlich und hilfsbereit. Sie ist auch auf der Suche
                    nach dem Schatz, der tief im Labyrinth verborgen sein soll.",
    "distance_to_player": 2.5,
    "gender": 1,
    "id": "npc_1",
    "is_following_player": true,
    "mood": "aufgeregt",
    "name": "Galadriel"
},
"player_position": "(3, 5)",
"target_position": "(1, 5)"
}
```

Über die Formatierung:

Verwende nur erlaubte Aktionen und deren benannte Parameter.

Gib deine Antwort ausschließlich als Liste von JSON-Objekten zurück.

Füge immer ein `"reason"`-Feld hinzu, um deine Entscheidung zu begründen.

Gib keinen Freitext oder sonstige Erklärungen außerhalb der JSON-Liste zurück.

**Antwort**

```
[
  {
    "action": "step_back:npc_1",
    "parameters": {},
    "reason": "Der Spieler bittet Galadriel darum, zurückzugehen, da er
              feststeckt. Ein Schritt zurück schafft direkt Raum."
  }
]
```

```
  },  
  {  
    "action": "talk:npc_1",  
    "parameters": {  
      "message": "Oh natürlich, Jan! Ich mache dir schnell etwas Platz.  
                  Sag Bescheid, falls du noch Hilfe brauchst."  
    },  
    "reason": "Galadriel reagiert hilfsbereit und freundlich auf die Bitte,  
              um die positive Stimmung beizubehalten und dem Spieler  
              Sicherheit zu vermitteln."  
  },  
  {  
    "action": "cheer:npc_1",  
    "parameters": {},  
    "reason": "Ein kurzer freundlicher Jubel lockert die Situation nach  
              der kleinen Hilfeleistung auf und stärkt die positive  
              Dynamik zwischen Galadriel und dem Spieler."  
  }  
]
```