Prof. Dr. Sturm, Spieleprogrammierung, Sommersemester 2025, Übungsblatt 2
Fabian Vecellio del Monego

Ergebnisbericht

1. Einleitung

Im Rahmen des Labyrinth-Projekts des ersten Übungsblatts wurde ursprünglich eine auf Basis einer `.txt`-Datei generierte 3D-Umgebung mit dynamischen Wänden, Kisten und Sammel-Objekten realisiert. Ziel dieser Erweiterung ist es, dem Labyrinth einen responsiven Non Player Charakter (NPC, „Quinn“) hinzuzufügen, dessen Verhalten nicht rein regelbasiert, sondern durch ein externes KI-Sprachmodell (Large Language Model, LLM) gesteuert wird.

Quinn nimmt ihre Umgebung über den PawnSensing-Component der Unreal Engine 5.4 wahr und entscheidet auf Basis verschiedener Umgebungsvariablen KI-gesteuert zwischen vier Grundverhaltensweisen:

- **Roaming**: Quinn erkundet das Labyrinth über randomisierte Pfade.
- **StayAndLook:** Quinn bleibt stehen und folgt dem Spieler mit ihrem Blick.
- **WalkAway:** Quinn entfernt sich moderat, sobald der Spieler sie belästigt.
- **RunAway:** Quinn flieht beschleunigt, wenn der Spieler sie beschießt.

Durch diese Architektur soll das NPC-Verhalten natürlicher, weniger deterministisch und damit menschlicher wirken. Die externe LLM-Anbindung erlaubt es, kontextabhängig Eskalationsstufen zu wählen und Deeskalation anhand von Distanz und vergangener Bedrohung intelligent vorzunehmen. Dieser Bericht dokumentiert die Motivation, die Software-Architektur, das Zustandsmodell von Quinn, die Implementierungsdetails sowie den Spielablauf und fasst abschließend Ergebnisse und gewonnene Erfahrungen zusammen. Im Anhang finden sich Diagramme zur Veranschaulichung meiner Implementierung. Außerdem habe ich das Verhalten des NPCs im Spiel in einem Video auf YouTube dokumentiert. Eine ausführliche Dokumentation zum Code findet sich in Form detaillierter Kommentare direkt in den C++-Dateien.

2. Technischer Kontext

In klassischen NPC-Implementierungen werden Verhalten und Eskalationsstufen meist fest im Code verankert (z. B. State-Machines mit harter `if`-Logik). Dieses regelbasierte Vorgehen skaliert jedoch schlecht, sobald komplexere Situationen (viele verschiedene NPCs, kombinierte Wahrnehmungskanäle, variable Eskalationsregeln, Deeskalation) abgebildet werden sollen. Bei der Anbindung eines externen LLM lassen sich Entscheidungsregeln dynamisch in Form von JSON-Kontext an das Modell übergeben. Die KI wertet auf Basis eines System Prompts fortlaufend gesammelte Wahrnehmungsdaten (Distanz, Sichtverbindung, Geschwindigkeit des Spielers, Waffenhaltung, Trefferhistorie, aktueller Zustand) aus und wählt daraus situativ die angemessenste Aktion.

Im vorliegenden Projekt nutzt die Klasse ANPCAgent die Unreal-eigene `PawnSensing`-Logik für Wahrnehmung und die `CharacterMovement`-Komponente für Fortbewegung. Die externen HTTP-Requests werden via VPN an das von Herr Bicanic auf einem Server in der Uni

Trier vorgehaltene LLM (Modell: *deepseek-coder-v2-lite-instruct*) geleitet. Realisiert wird das über das HTTP-Module der UE 5. Dadurch bleibt die Engine-Loop frei von blockierenden Operationen, und das NPC-Verhalten passt sich dynamisch jeder Spielsituation an.

3. Software-Architektur

Meine Implementierung baut auf dem zu Übungsblatt 1 erstellten Projekt auf. Basis ist also ein in First-Person-Perspektive begehbares Labyrinth, in dem Spieler neben prozedural platzierten Kisten und Waffen einen Schatz finden können, um das Level mit anders platzierten Elementen neu starten können. In jenes Level habe ich jetzt einen NPC-Charakter integriert, den ich weitestgehend dem Third-Person-Template der Unreal Engine 5 entliehen habe. Die Engine bietet auch entsprechende Assets für das Skelett, die Texturen und die Animationen, auf die ich entsprechend zurückgegriffen habe.

Die Logik für unter anderem die Navigation des NPCs, die Hilfsmethoden zur Bestimmung des Spielerverhaltens, die Konstruktion der LLM-Requests, deren Auswertung und Umsetzung sowie die Verbindung zum LLM-Server habe ich in einer neuen C++-Klasse gebündelt, `ANPCAgent`. In Abbildung 1 (siehe Anhang) ist die statische Struktur der zentralen Klassen als UML-Klassendiagramm dargestellt.

Der Ablauf der NPC-Logik gliedert sich in den Aufbau des Chat-Payloads mit System- und User-Nachricht (JSON-Serialisierung), einen Asynchronen POST-Request an den LLM-Endpoint und den Callback `OnAIResponse()`. Damit einher gehen eine Prüfung auf HTTP-Erfolg und gültiges JSON-Format, die Extraktion des Aktions-Strings, die Zuweisung von `CurrentAction` und das Auslösen der Funktionen wie `StopMovement()`, `SimpleMoveToLocation()` und weiterer Logik. Zusätzlich wird in jedem Schritt + geprüft, ob die Antwort valide ist (Fehlercodes, fehlende Felder) und bei Bedarf ein erneuter Request später erlaubt. Der Ablauf folgt dem Sequenzdiagramm in Abbildung 2 (siehe Anhang).

4. Verhaltenslogik des NPC

Der Kern der Quinn-Logik ist grundsätzlich nicht deterministisch, lässt aber in seiner Intention als **State Machine** mit vier wesentlichen Zuständen begreifen:

1. **Roaming**: Quinn wandert zufällig durch das Labyrinth, solange der Spieler weder in Sichtweite ist noch Schaden verursacht wird.
2. **StayAndLook**: Quinn bleibt stehen und richtet ihren Blick kontinuierlich auf den Spieler. Dieser Zustand tritt für gewöhnlich ein, wenn Quinn den Spieler sieht, der Spieler aber weder zu nahe kommt noch schießt. Auch die Geschwindigkeit des Spielers kann eine Rolle spielen. Ebenso erhält Quinn die Information, ob der Spieler auf sie zielt.
3. **WalkAway**: Quinn entfernt sich potenziell mit moderater Geschwindigkeit, sobald der Spieler als Bedrohung wahrgenommen wird. Das kann beispielsweise durch eine geringe Distanz, ein Anrempeln, ein schnelles Zulaufen oder das Zielen mit der Waffe begründet sein. Außerdem ist es möglich, dass Quinn den Spieler meidet, wenn sie in der Vergangenheit belästigt oder beschossen wurde.
4. **RunAway**: Quinn flieht tendenziell mit hoher Geschwindigkeit, wenn der Spieler sie beschießt oder sie dem Spieler begegnet und diese sie in junger Vergangenheit beschossen hat.

Eindeutige Übergangsregeln lassen sich bei System Prompts ohne strenge Anweisungen nicht formulieren. Auf diesen Umstand gehe ich im nächsten Kapitel im Detail ein.

5. Details der Implementierung

Das tatsächliche Verhalten des LLM und damit des NPC hängt maßgeblich vom System Prompt und den zur Verfügung gestellten Informationen ab. In einer ersten Iteration übergab ich dem LLM nur Variablen zu Distanz und Sichtkontakt und bat um eine Entscheidung zwischen zwei zur Auswahl stehenden Strings, die es als Antwort liefern sollte. Eine konstruktive Entscheidungsfindung war damit aber nur eingeschränkt möglich. Ohne Information über Treffer oder den aktuellen Zustand konnte das LLM nicht zuverlässig erkennen, wann wirklich Gefahr bestand. Sukzessive habe ich daher mehr Daten vorgehalten und dem Prompt im JSON-Format beigefügt.

Entscheidend wurden damit immer mehr der Wortlaut des System Prompts und die Leistungsfähigkeit des LLM. Weil ich mich im Rahmen dieser Implementierung auf den von Ihnen zur Verfügung stehenden Server mit einer DeepSeek-Instanz entschieden habe, blieb insbesondere der Prompt als freie Variable.

Hier habe ich die Erfahrung gemacht, dass es sehr schwierig ist, dem LLM einerseits weitgehende Freiheit in seiner Entscheidung respektive Interpretation der Bedrohungslage zu geben und andererseits einen verlässlichen und nachvollziehbaren Wechsel zwischen allen vier zur Verfügung stehenden Zuständen zu gewährleisten. Je nachdem, wie sehr ich im System Prompt vor dem Spieler gewarnt habe oder zur deeskalierenden Koexistenz gemahnt habe, wechselte das LLM zwar zwischen den ersten beiden oder letzten beiden Eskalationsstufen, sehr selten aber zwischen allen drei. Häufig war es so, dass das LLM nur noch wegrennen wollte, sobald einmal eine Bedrohung ausgemacht war; ganz gleich wie lange das her war. Oder aber das LLM wollte partout nie fliehen, sondern ließ auch einen Kugelhagel geduldig über sich ergehen.

Mit sukzessive erweitertem Informationsfluss stellte sich diesbezüglich eine Besserung ein, in Kombination mit der nichtdeterministischen Natur der KI habe ich mich aber zu Demonstrationszwecken für einen System Prompt entschieden, der das LLM strikt an die Hand nimmt, um das in Abbildung 3 (siehe Anhang) gezeigte Zustandsdiagramm zu gewährleisten. Ein Beispiel für einen Prompt, der dem LLM wiederum einen größeren Interpretationsspielraum einräumt und damit aus Perspektive des Spielers durchaus auch interessantere, weil dynamischere, Ergebnisse liefern kann, sieht derweil wie folgt aus:

```
"You are Quinn, an NPC in a 3D labyrinth. "
"There is a human player in the same maze with you. "
"He may prove hostile but assume that he is friendly in the first place. "
"You may encounter him. If so, you can choose to greet him by staying and
looking at him. "
"It is possible, however, that he runs into you, pushing you around, or
even starts to shoot at you. "
"In this case, you may walk or even run away. "
"Your current situation is represented by the following JSON file:\n"
"  • distanceCm (number): distance to the player in cm\n"
"  • hasLineOfSight (bool)\n"
"  • isAimingAtYou (bool)\n"
"  • isPlayerRunningTowardsYou (bool)\n"
"  • recentHits (int): number of times the player shot at you\n"
"  • timeSinceLastHit (number): seconds since the last hit\n"
```

```
"  • currentAction (string): one of
[\"StayAndLook\",\"WalkAway\",\"RunAway\"]\n\n"
"Based on the given information, please advise Quinn’s next action. "
"Reply with exactly one of: StayAndLook, WalkAway, RunAway."
```

Bei jedem Sicht-Event (über PawnSensing) oder Treffer-Event (in TakeDamage) neue HTTP-Requests abzusetzen, wäre extrem ineffizient und teuer. Deshalb habe ich zwei separate Intervalle definiert: Normale Requests (durch Sichtkontakt) sind maximal einmal alle 3 Sekunden erlaubt und Hit-Requests (durch Schaden) maximal einmal alle 2 Sekunden. Intern speichere ich dazu zwei Zeitstempel, `LastAIRequestTime` und `LastHitAIRequestTime`. Jede neue Anfrage wird nur dann losgeschickt, wenn der jeweilige Mindestabstand zur letzten Anfrage eingehalten ist.

So bleibt Quinn reaktionsfähig bei Bedrohung, ohne in einer Flut von Netzwerkaufrufen unterzugehen. Das hat außerdem den Vorteil, dass Quinn nicht zu häufig zwischen verschiedenen Zuständen hin- und herspringt und damit für den Spieler lesbar bleibt.

6. Ablauf im Spiel

In einem Demo-Video zeige ich den vollständigen Ablauf eines Spieldurchlaufs mit mehreren Begegnungen mit Quinn im Labyrinth. Der Vollständigkeit halber habe ich nachfolgend auch im Bericht eine beispielhafte Begegnung geschildert.

1. Quinn wandert frei umher. Sofern sie den Spieler nicht sieht oder von ihm berührt bzw. beschossen wird, hält dieser Zustand an.
2. Der Spieler tritt in Quinns Sichtfeld. Sie entscheidet sich, ihn zu grüßen, was sie mit Stehenbleiben und Ansehen umsetzt.
3. Der Spieler läuft auf sie zu und bleibt unmittelbar vor ihr stehen. Quinn fühlt sich bedrängt und geht einige Schritte weg, bevor sie wieder beginnt, das Labyrinth frei zu erkunden.
4. Der Spieler verfolgt sie und rempelt sie an. Sie geht erneut weg.
5. Sie dreht sich und sieht den Spieler. Aufgrund der kürzlichen negativen Erfahrungen bleibt sie nicht stehen, sondern geht weiter weg.
6. Der Spieler sammelt eine Waffe auf und schießt auf Quinn. Sie beginnt wegzurennen.
7. Der Spieler verfolgt sie und beschießt sie kontinuierlich. Sie rennt weiter.
8. Der Spieler bleibt stehen und lässt einige Sekunden verstreichen. Quinn bleibt stehen und beginnt erneut die freie Erkundung des Labyrinths.
9. Quinn sieht den Spieler. Aufgrund des kürzlichen Beschusses rennt sie wieder weg.
10. Es vergehen einige Minuten. Quinn sieht den Spieler erneut. Sie geht weg, rennt aber nicht.
11. Es vergehen noch einmal einige Minuten, in denen der Spieler Quinn nicht belästigt hat. Beim erneuten Aufeinandertreffen bleibt sie grüßend stehen.

7. Fazit

Die Bearbeitung des 2. Übungsblatts mit Quinns KI-Verhalten hat mich insgesamt etwa 35 Stunden beschäftigt und war dabei zwar lehrreich, aber auch (zeitlich) anspruchsvoll, auch im Vergleich zu bisherigen Abgaben bei Ihnen. Die Unreal Engine 5.4 mit ihrem PawnSensing-System, den SpringArm-Komponenten und den Animation-Blueprints bot eine solide Grundlage, sodass zumindest grundlegendes Roaming und Blick-Tracking im Labyrinth rasch

implementiert werden konnten. Zwar fehlte mir zwischendurch oft der volle Durchblick in die Engine-Interna, aber dank der zahlreichen Online-Dokumentationen und Forenbeiträge lief der technische Teil zur Implementierung der NPC-Hülle letztlich recht geschmeidig.

Der spannendste Teil der Umsetzung lag dann in der Integration des externen LLM. Herr Bicanics Server stellte eine unkomplizierte Basis bereit, sodass schnell mit ersten Prompts außerhalb der Engine experimentiert werden konnte. Nach kurzer Zeit lief dann auch der erste HTTP-Request über das Unreal-HTTP-Modul.

Herausfordernd war es dann, beides zusammenzufügen und sinnvoll in meine bestehende Labyrinth-Implementierung einzubinden. Dabei gab es zwei Hürden: Einerseits war es ohnehin nicht trivial, den NPC über Strings zu steuern. Die Logik für die Einbindung des LLM sowie den damit einhergehenden Overhead ist sehr umfangreich und hat mich viele Stunden beschäftigt, bis das Vorhaben zufriedenstellend lief. Die zweite Hürde war dann die nichtdeterministische Natur der KI. Die Prompts in einem Wortlaut zu formulieren, der einerseits einen Mehrwert gegenüber verschachtelten Fallunterscheidungen im Code bietet und andererseits zu einem nachvollziehbaren wie aus Sicht des Spielers lesbaren Verhalten Quinns führt, war nicht einfach und ohne Kompromisse möglich.

Letztlich hat das Projekt aber zum Ende einen Zustand erreicht, der nicht nur den Anforderungen des Übungsblatts gerecht wird, sondern im Spiel auch tatsächlich Spaß macht und einen Mehrwert bietet. Zwar lässt sich das Verhalten der KI nach wie vor als deterministische State Machine approximieren und in dieser Art und Weise sicherlich über klassische Fallunterscheidungen mit einer Heuristik und Zufallselementen nachahmen, als Machbarkeitsstudie ist das Ergebnis aber sehenswert und vermittelt einen Eindruck des Potenzials KI-gesteuerter NPCs. LLMs sind dabei meiner Meinung nach zwar als Universalwerkzeug ein möglicher, aufgrund ihres Overheads und Umwegs über natürliche Sprache nicht aber der optimale Weg. Lokal eingebettete und dedizierte Machine-Learning-Ansätze sind diesbezüglich aber sicherlich vielversprechend.

Anhang

Die nachfolgenden 3 Diagramme habe ich dieser Abgabe auch als hochauflösende PNG- bzw. SVG-Dateien beigefügt.

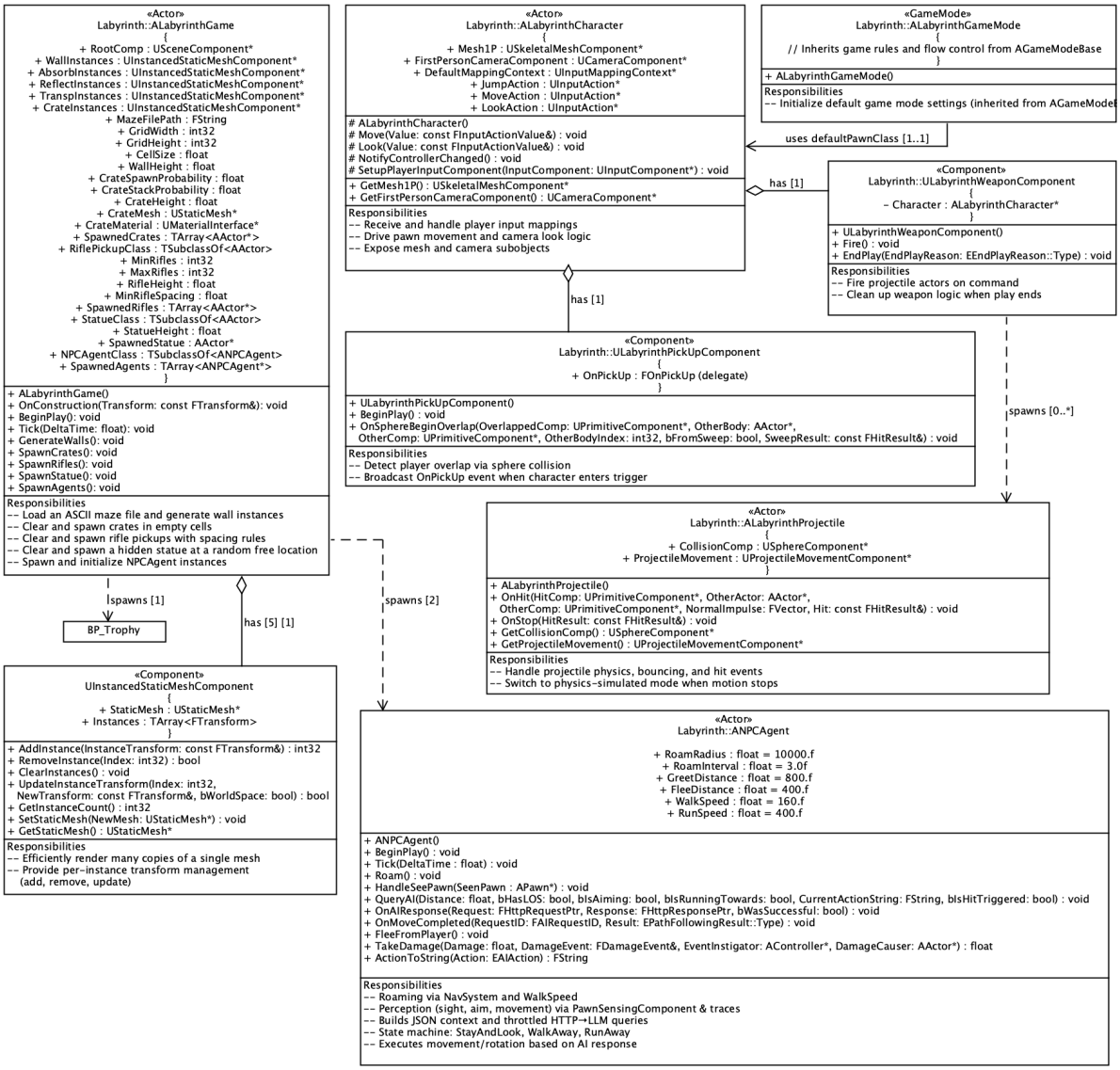

Abbildung 1: UML-Klassendiagramm

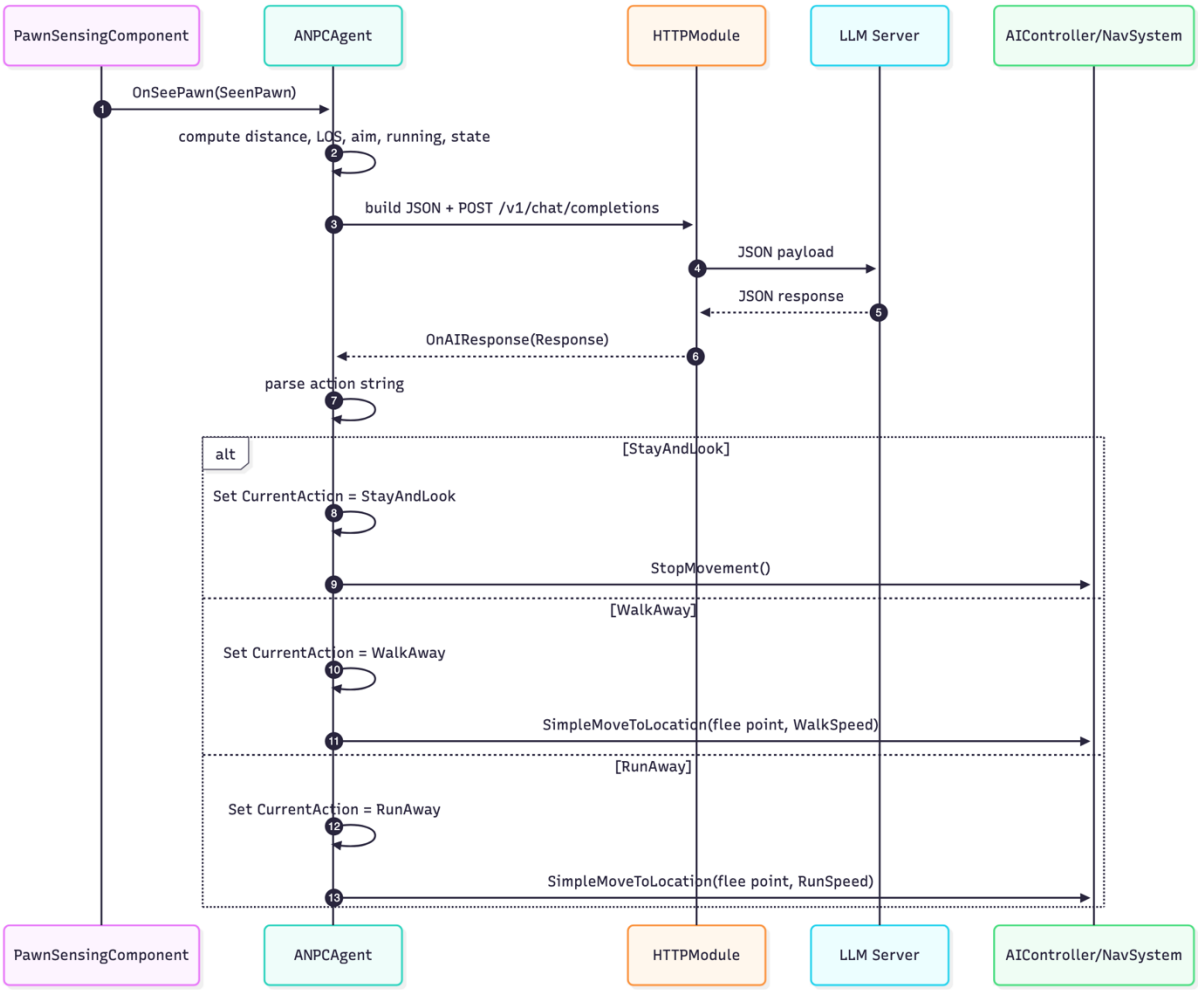

Abbildung 2: Sequenzdiagramm

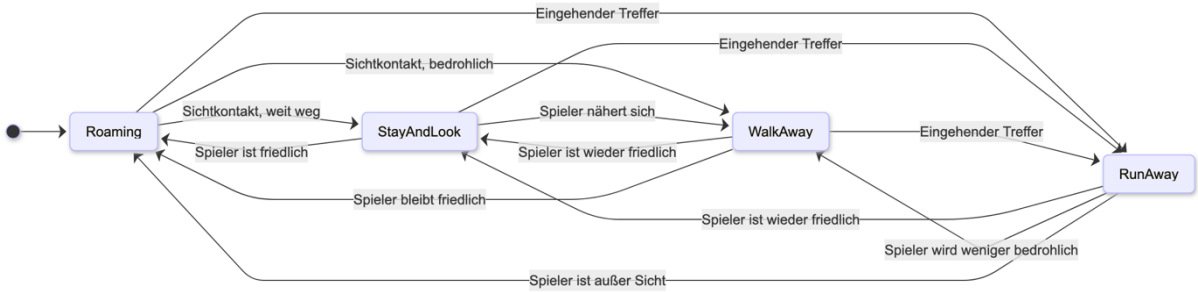

Abbildung 3: Zustandsdiagramm (approximierte „NPC State Machine“)