Hochschule für Technik und Wirtschaft Berlin

Fachbereich 4

Entwicklung einer taktischen Squad-K.I. mit Debug-Visualisierung in Unity

Bachelorarbeit zur Erlangung des akademischen Grades Bachelor of Science,
Internationaler Studiengang Medieninformatik

vorgelegt von Bernard Zaborniak

Matrikelnummer 558930

am 20.03.2021

Erstprüfer Prof. Dr. Tobias Lenz

Zweitprüfer Prof. Dr.-Ing. David Strippgen

Inhaltsverzeichnis

1 Einleitung

1.1 Problemstellung

Die Künstliche Intelligenz (KI) kann entscheidend für die Erfahrung in einem Spiel sein. Abhängig von der Funktionsweise kann die KI den Spielspaß verbessern und neue Gameplay-Möglichkeiten öffnen oder diesen verschlechtern und die Immersion brechen. Außerdem stellt die KI für Entwickler eine große Herausforderung dar. So kann die Performance einer KI die Anzahl der Charaktere im Spiel stark beschränken. Auch kann es für Entwickler schwierig sein, die Entscheidungen einer KI nachzuvollziehen, da diese oft nur begrenzt visuell oder akustisch wahrnehmbar sind, wodurch die Entwicklungszeit sich verlängern kann.

1.2 Zielsetzung

Die Bachelorarbeit soll sich mit der Planung und Entwicklung einer Künstlichen Intelligenz für einen 3D first-person Shooter befassen. Hierbei sollen sich jeweils Gruppen von menschlichen Soldaten (im Folgenden Squads genannt) gegen andere Squads oder einen einzelnen Spielercharakter ein Feuergefecht liefern und dabei sinnvolle taktische Entscheidungen treffen. Dabei soll ein besonderer Wert auf eine verständliche Visualisierung der durch die K.I. getroffenen Entscheidungen gelegt werden. Diese Visualisierung ist nicht für den Spieler, sondern für den Entwickler gedacht und soll dazu dienen die Entwicklung der KI zu vereinfachen und zu beschleunigen.

Neben der schriftlichen Arbeit wird mit der Unity Engine ein Prototyp der KI in einer 3D-Umgebung erstellt.

2 Konzept

Im Folgenden wird das Konzept für die KI-Architektur und alle dafür benötigten Bausteine erläutert. Das Projekt besteht aus insgesamt 5 Teilen, welche alle in Reihenfolge entwickelt wurden und jeweils aufeinander aufbauen. Diese Teile nennen sich *Character Controller*, *Abstract World Representation*, *Individual AI* und *Squad AI.*

2.1 Begriffserklärung

2.1.1 Agent

Als Agent wird in der folgenden Arbeit ein Charakter bezeichnet, der mit einer KI ausgestattet ist, sich in der Spielwelt befindet und dort agiert.

2.1.2 Aktionen

Ein Agent kann bestimmte Aktionen ausführen. Diese sind für den Spieler als Animationen oder Bewegungen wahrnehmbar oder als Geräusche hörbar.

2.1.3 Künstliche Intelligenz

In der folgenden Arbeit wird als Künstliche Intelligenz der Algorithmus bezeichnet, der dafür zuständig ist, dass ein Agent abhängig von der aktuellen Situation in der Spielwelt stets eine passende Aktion ausführt. Dadurch soll dieser Algorithmus für den Spieler die Illusionen einer Intelligenz erzeugen.

2.1.4 Szene/Spielwelt

Als Szene wird in Unity eine Spielewelt bezeichnet. Sie beinhaltet im Spiel gesehenen Objekte wie Umgebungen oder Charaktere. Jede Szene hat einen Namen und eine Szenenhierarchie mit Objekten.

2.2 Character Controller

Um Charaktere in der Spielwelt mit einer KI auszustatten, bedarf es erstmal der Charaktere selbst. Diese sollten sich in der Spielwelt fortbewegen und mit anderen Charakteren oder Objekten interagieren können. Das System, welches sich um die Bewegung und Animation der Charakter kümmert, nennt sich im Projekt *Character Controller*. Wenn das Äquivalent zu der KI der Charaktere in der echten Welt das Gehirn eines Menschen wäre, kann man sich den *Character Controller* als Nervensystem vorstellen. Dieses wartet auf Befehle vom Gehirn (KI) und führt diese als Bewegungen aus.

Ich habe mich dafür entschieden, menschliche Charaktere zu nutzen und diese mit realistischen Animationen auszustatten. Dies war eine riskante Entscheidung, da die Animation eines menschlichen Körpers komplex und daher auch fehleranfällig ist. Außerdem könnte ein komplexes Animationssystem das Testen der KI erschweren, da ein Entwickler und vor allem ein Spieler bei unerwartetem Verhalten schwer einschätzen kann, ob der dazu führende Fehler bei der KI oder bei dem Animationssystem liegt. Unrealistisch aussehende Animationen können dazu führen, dass ein Charakter vom Spieler als “dumm” eingestuft wird, obwohl die Entscheidung der KI durchaus nachvollziehbar wäre.

Im Gegensatz dazu können gute Animationen unpassende Entscheidungen der KI vertuschen und zur Immersion beitragen.

2.2.1 Anforderungen an den Character Controller

Der *Character Controller* soll folgende Aktionen durchführen können:

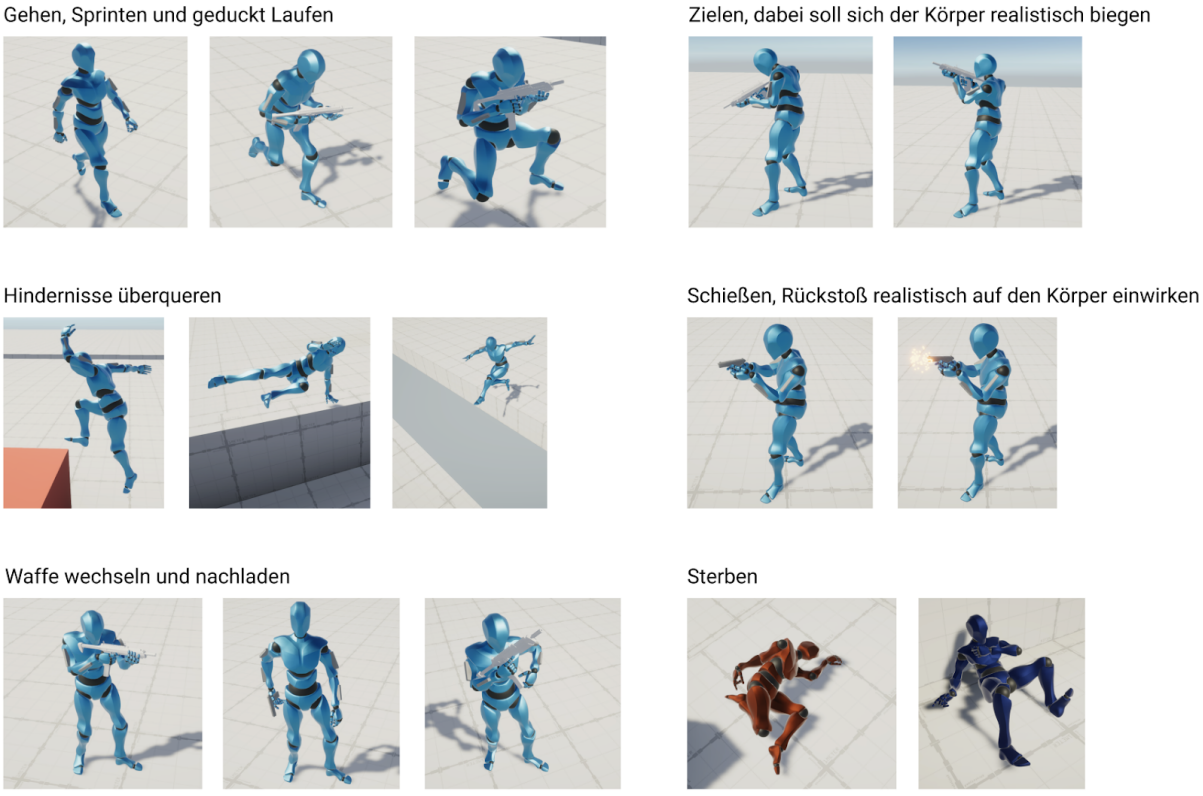


Abbildung 1: Anforderungen an den Character Controller

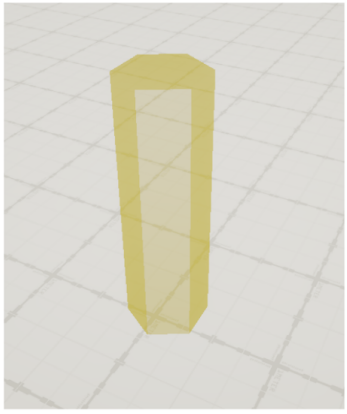
2.3 Abstract World Representation

Menschen treffen Entscheidungen basierend auf Informationen, die sie der Umwelt entnehmen. Unsere Gehirne können die verschiedenen Formen und Farben, die uns unsere Augen zusenden, beispielsweise als Text oder Gesichter anderer Menschen deuten. Eine Spiele-KI könnte die Welt auf ähnliche Weise wahrnehmen. Eine Kamera würde die Welt aus der Sicht des Charakters in der Spielewelt jeweils mithilfe einer begrenzten Anzahl an Pixeln als Bildinformation speichern. Die KI kann dieses Bild analysieren und darauf basierend, Entscheidungen treffen.

Dies wäre aber ein überaus komplexes Unterfangen. Ähnliche Ergebnisse lassen sich dadurch erzielen, dass komplexe Informationen in abstrakter Form in einer Spielwelt gespeichert werden. Diese sollen für die KI leicht lesbar sein. Beispielsweise kann eine Position, welche eine gute Deckung darstellt, in der Spielewelt markiert sein. Gespeichert wären die Informationen über Position und Bewertung dieser Deckung. Auch wir Menschen nehmen wichtige Informationen über unsere Umgebung viel schneller und einfacher wahr, wenn sie durch bspw. leicht lesbare Schilder markiert sind, sei es ein Notausgang oder ein Feuerlöscher.

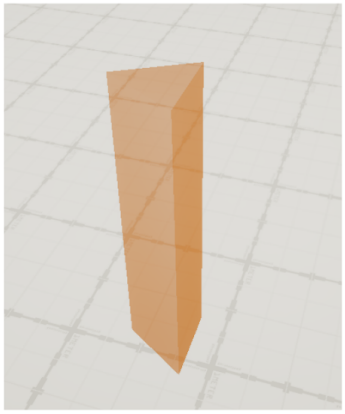
2.3.1 Tactical Points

In der Spielwelt werden vom Entwickler Objekte platziert, welche Informationen über die Deckung an einem bestimmten Ort liefern sollen. Diese Objekte werden *Tactical Points* genannt. Die *Points* können in der Welt vom Entwickler platziert oder prozedural generiert werden.



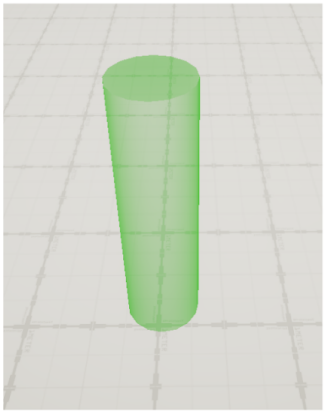
Cover Point

Markiert einen Punkt, wo zumindest in eine Richtung, Schutz vor feindlichen Schüssen gegeben ist.



Cover Peek Point

Kommt nur zusammen mit einem Cover Point vor. Die Peek Points markieren Positionen, wo aus der Deckung raus geschossen werden kann.



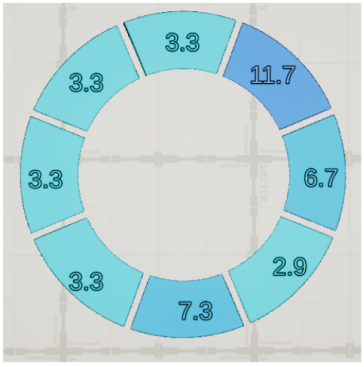
Open Field Point

Diese Punkte beschreiben die Deckung an beliebigen anderen Orten. Sie können bei der Wegefindung von Vorteil sein.

Abbildung 2: Tactical Points

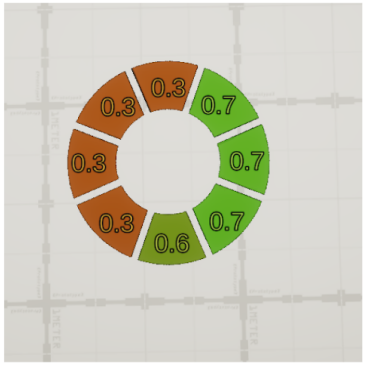
Inspiriert vom Paper **“Killzone’s AI: dynamic procedural combat tactics”**, werden Informationen über die Deckung in jeweils 8 Richtungen gespeichert. Jede dieser Richtungen besitzt 2 Werte.

Durch das Nutzen verschiedener Farben, lassen sich diese Bewertung auf verständlicher Weise visualisieren. Da der Charakter jeweils knien oder stehen kann, gibt es für beide Höhen verschiedene numerische Bewertungen der Deckung.



Distance Rating

Diese 8 Werte informieren darüber, wie weit eine Deckung in die jeweilige Richtung entfernt ist.



Quality Rating

Diese 8 Werte beschreiben die Qualität der Deckung auf der vorher erwähnten Distanz auf einer Skala 0-1.

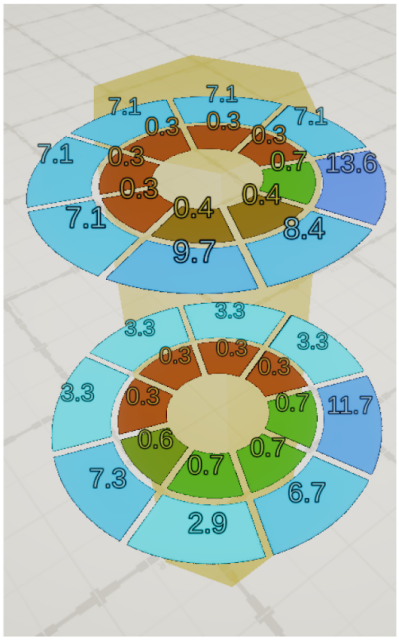


Abbildung 3: Distance & Quality Rating

2.4 Sensing

Bevor eine KI entscheidet, welche Aktion sie ausführen soll, benötigt sie Informationen. Diese Informationen müssen der Spielewelt entnommen und in einer Art Gedächtnis gespeichert werden.

Um diese Informationen zu erlangen, soll der Charakter seine Umgebung in einem bestimmten Radius scannen und dabei Informationen über gegnerische oder verbündete Charaktere und *Tactical Points* speichern.

2.4.1 Blackboard

Die über die Umgebung gewonnen Informationen werden im *Blackboard* gespeichert. Dieses System speichert die Informationen aus allen vorherigen Umgebungs-Scans mit der dazugehörigen Zeit. Ältere Informationen werden erst überschrieben, sobald das *Sensing* dem *Blackboard* aktuellere Informationen zu demselben Objekt liefert. Außerdem speichert das *Blackboard* andere für die KI relevante Informationen, wie beispielsweise den Zustand des Charakters oder seiner Waffen.

2.5 Individual AI

Jeder Charakter in der Spielwelt hat seine eigene KI, welche im Projekt *Individual AI* genannt wird. Diese *Individual AI* entscheidet basierend auf den Informationen vom Sensing darüber, welche Befehle sie dem *Character Controller* schickt, um den Charakter zu bewegen und zu animieren.

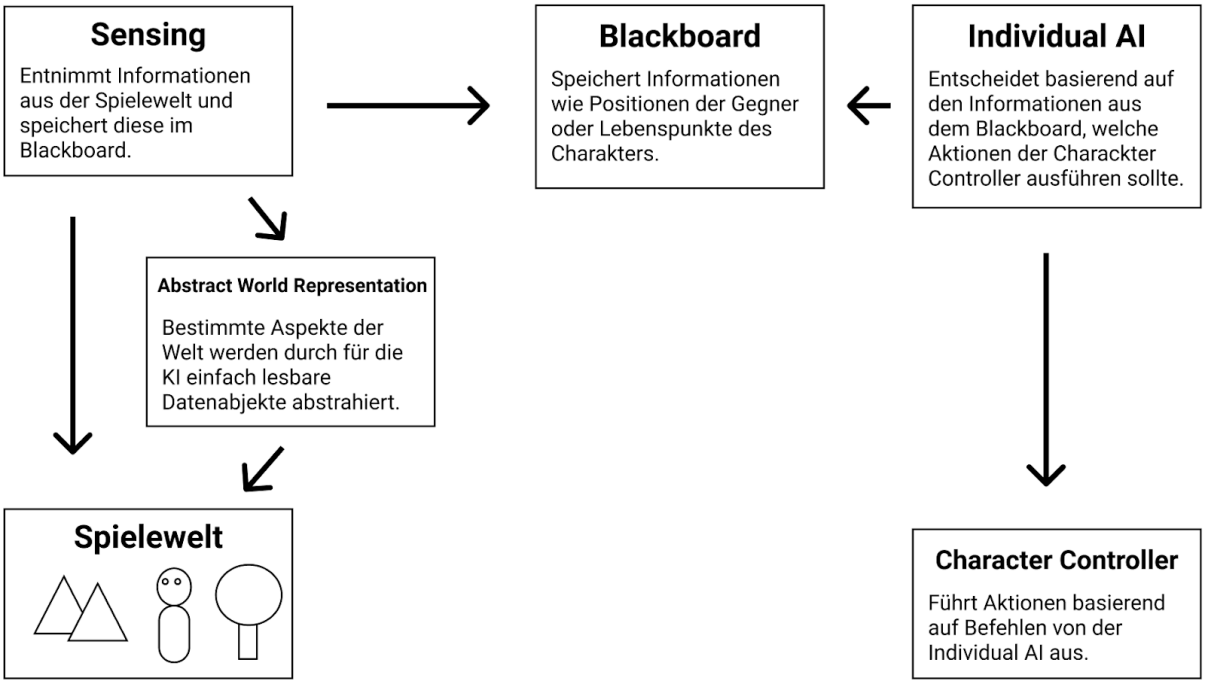


Abbildung 4: Kommunikation mit der Individual AI

In der Spieleindustrie haben sich über die letzten Jahre hinweg, zahlreiche Standards für die Implementierung einer KI für individuelle Agenten etabliert. Im Grunde sich alle derzeit üblichen KI-Architekturen sehr ähnlich. In den meisten Architekturen geht es darum, aus einem vordefinierten Set an Aktionen, jeweils die passendste auszuwählen und auszuführen. Der Unterschied liegt im Entscheidungsprozess. Eine Auswahl sollte in Abhängigkeit davon getroffen werden, was für ein Spiel entwickelt wird, wie viel Kontrolle ein Entwickler über die Entscheidungen der KI besitzen möchte und wie viel Zeit zur Verfügung steht.

Ich entschied mich für das “Infinite Axis Utility System”, eine Variation eines “Utility-based” Systems, welche von **Dave Mark** in zahlreichen Vorträgen auf der jährlichen **Games Developer Conference** vorgestellt und weiterentwickelt wurde. Das von mir vorgestellte Konzept für die individuelle KI ist sehr stark von dem von Dave Mark vorgestelltem Konzept inspiriert.

Die Grundlage eines “Utility-based”, oder zu Deutsch “Nutzen-basierten Systems”, ist die numerische Bewertung aller der KI zu Verfügung stehenden Aktionen basierend auf Ihrem aktuellen Nutzen für den Charakter. Die Bewertung einer Aktion kann auf verschiedene Weisen stattfinden und ist fundamental für den Entscheidungsprozess.

2.5.1 Entscheidung in einem Utility-based System

In der folgenden Visualisierung ist ein vereinfachtes Beispiel eines solchen Entscheidungsprozesses zu sehen. Ein Charakter befindet sich in einem Raum und muss sich für eine Aktion, basierend auf seinen Bedürfnissen entscheiden.

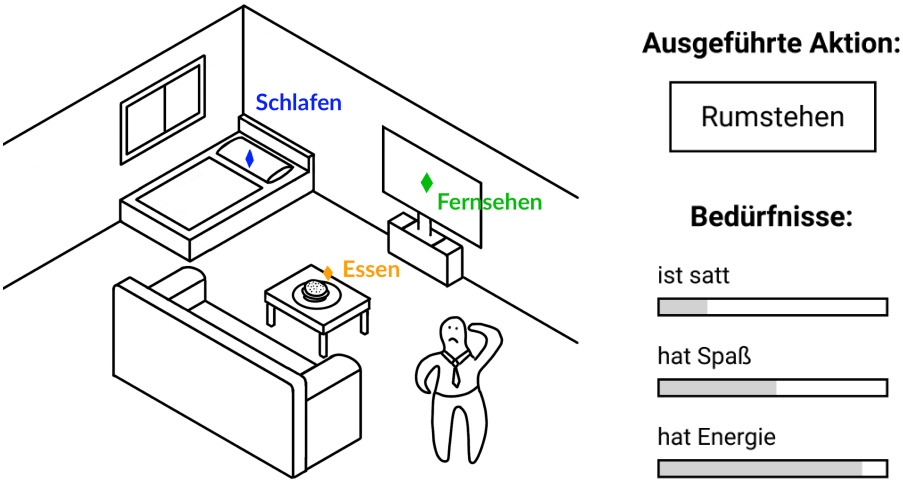


Abbildung 5: Entscheidungen in Utility-based System 1

Die zur Verfügung stehenden Aktionen werden basierend auf den Bedürfnissen mit Zahlen von 0 bis 1 bewertet. Die am besten bewertete Aktion wird ausgeführt.

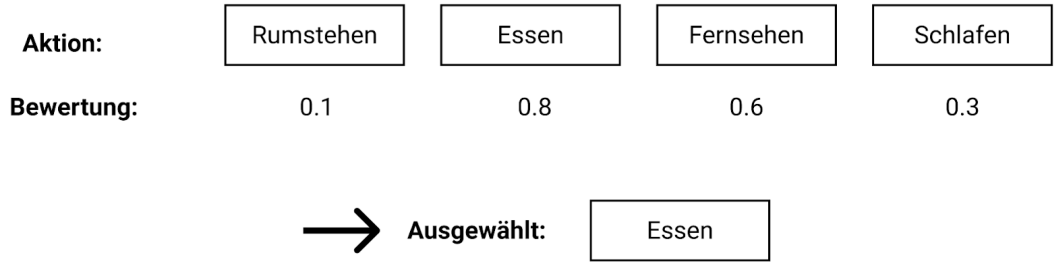


Abbildung 6: Entscheidungen in Utility-based System 2

In diesem Fall wurde die Aktion Essen ausgewählt, da das Bedürfnis satt zu werden am größten war. Die Aktion Schlafengehen wurde mit einer viel geringeren Bewertung versehen, da der Energiebalken des Charakters aktuell fast komplett gefüllt ist.

Nachdem der Charakter gegessen hat und satt ist, haben sich seine Bedürfnisse geändert. In der nächsten Entscheidungsphase würde die Aktion Schlafen gewählt werden, da nun der Energiebalken fast leer ist.

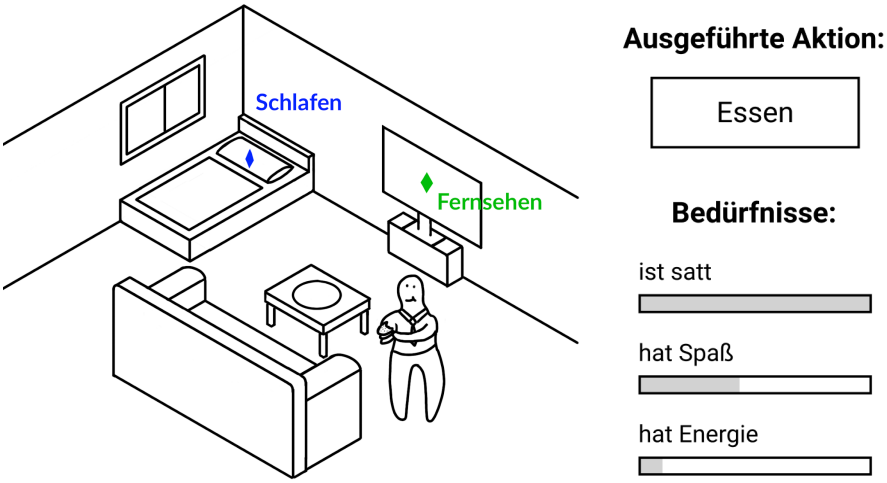


Abbildung 7: Entscheidungen in Utility-based System 3

2.5.2 Infinite Axis Utility System (IAUS)

Im Folgenden wird meine Interpretation des *IAUS* vorgestellt, welches auf dem gerade vorgestellten Utility-based System basiert. Die meisten der im Folgenden vorgestellten Komponenten und Zusammenhänge wurden dem Vortrag **"Building a Better Centaur: AI at Massive Scale"** auf der Game Developers Conference 2015 entnommen. Dieser wurde von Mike Lewis und Dave Mark gehalten.

Decisions

Das Fundament des *IAUS* ist eine *Decision*. Auf Deutsch wäre diese Komponente als Entscheidung zu Bezeichnen. Ein *Decision* Objekt beinhaltet Informationen darüber, welche Aktion ausgeführt werden soll, welches Ziel diese Aktion haben kann und unter welchen Umständen diese Aktion ausgeführt werden soll.

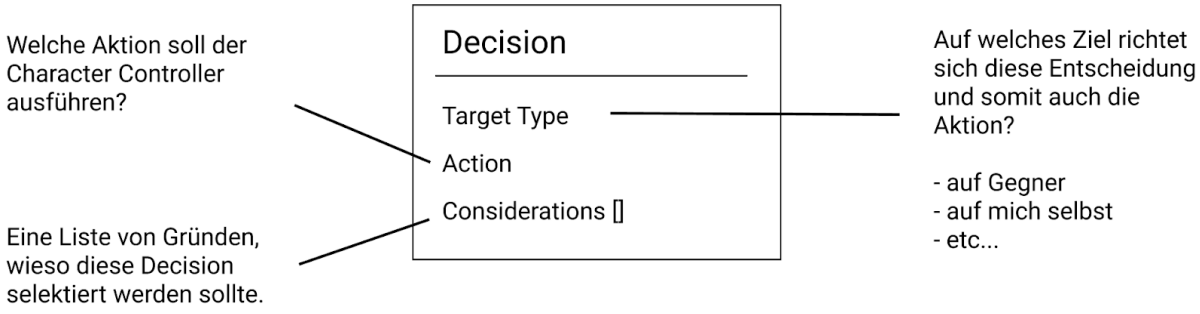


Abbildung 8: Grundkomponenten einer Decision

Decision Maker

Eine KI besitzt ein oder mehrere *Decision Maker*. Ein *Decision Maker* besitzt eine Liste an *Decisions*, die der KI zur Verfügung stehen. In einem festgelegten zeitlichen Abstand werden alle *Decisions* mit Zahlen bewertet und die Aktion der am besten bewerteten *Decision* wird ausgeführt. *Decisions* welche sich auf andere Ziele als den Charakter selbst beziehen, werden für jeden dieser Ziele separat bewertet.

Beispielsweise existiert die *Decision* namens “Schieße auf den Gegner”. Diese *Decision* wird für alle zur Verfügung stehenden Gegner separat bewertet. Da sich Gegner durch Lebenspunkte oder Entfernung zum entscheidenden Charakter unterschieden, werden sich auch die Bewertungen der *Decisions*, auf die verschiedenen Gegner zu schießen, unterscheiden.

Die KI würde sich in der *Abbildung 9* entscheiden, auf den Gegner 2 zu schießen, da diese *Decision* am besten bewertet wurde.

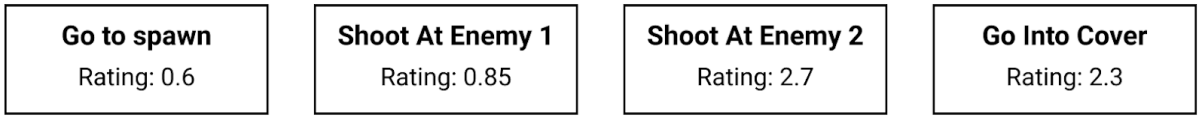


Abbildung 9: Bewertete Decisions

Considerations

Die *Considerations* sind die Schnittstelle zwischen dem ***Blackboard*** und dem Entscheidungsprozess. Auf Deutsch könnten diese als Erwägungen übersetzt werden. Dies sind die Grundbausteine, durch welche eine *Decision* bewertet wird. Ein *Consideration* Objekt liest eine Information und bewertet diese mit einem Wert von 0 bis 1. Anders formuliert, könnten *Consideration* als Gründe bezeichnet werden, wieso eine Aktion ausgeführt werden sollte. Diese Gründe können auf einer Skala von 0 bis 1 mehr oder weniger zutreffen.

Bewertung einer Decision durch die Considerations

Um eine *Decision* zu bewerten, wird die komplette Liste an *Considerations* bewertet und miteinander und mit dem *weight* einer *Decision* multipliziert. Der *weight* ist eine Zahl, die zum Priorisieren von *Decisions* genutzt wird.

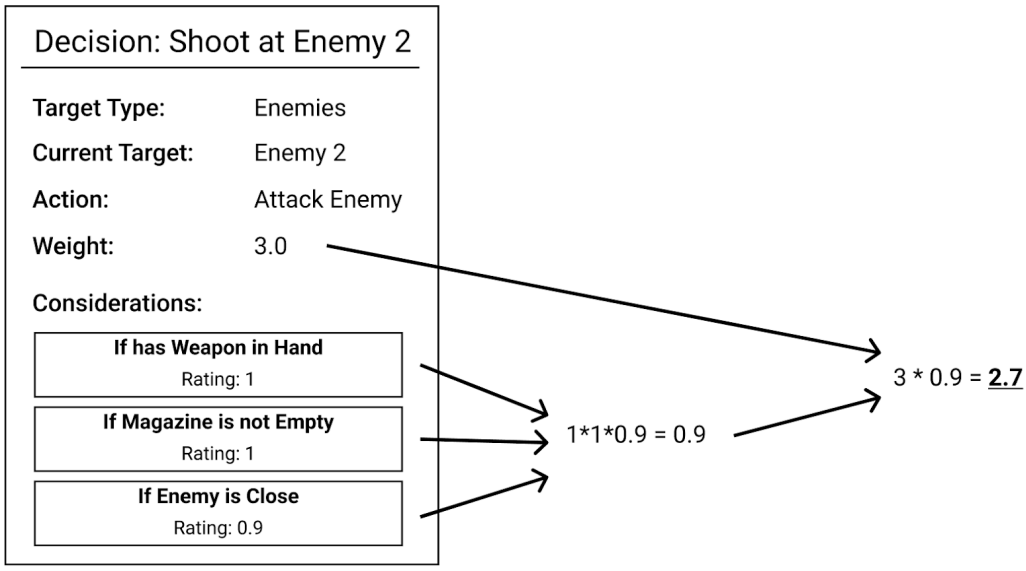


Abbildung 10: Bewertung einer Decision

Makeup Value

Ein Problem bei dieser Methode wäre, dass die Bewertung bei *Decisions* mit vielen *Considerations* oft viel kleiner ausfallen würde. Beispielsweise hätte eine *Decisions* mit 8 *Considerations*, von denen jede eine Bewertung von 0.9 hätte, einen Score von nur 0.43. Eine *Decision* mit 2 *Considerations* hingegen, von denen jede mit 0.8 bewertet wäre, hätte einen Score von 0.64.

Dave Mark schlägt in der Vorlesung **Building a Better Centaur: AI at Massive Scale** auf der Game Developers Conference 2015 vor, diesem Prozess mit einem *Makeup Value* entgegenzuwirken. Der *Makeup Value* wird mit der folgenden Formel berechnet und zu dem Produkt der *Consideration*-Bewertungen addiert. Das Produkt der *Consideration*-Bewertungen wird in der Formel als *score* bezeichnet.

$$\text{Makup-Value} = \text{score} * ((1 - \text{score}) * (1 - 1/\text{numberOfConsiderations}))$$

Bewertung einer Consideration

Im Folgenden ist ein Beispiel der *Consideration* mit dem Namen “Bin ich nah am Gegner” abgebildet. Die Bewertung dieser *Consideration* soll das Gefühl der damit verbundenen Gefahr widerspiegeln. Damit verschiedene *Considerations* miteinander vergleichbar sind, wird der Input jeweils anhand festgelegter Minimal- und Maximalwerte normalisiert.

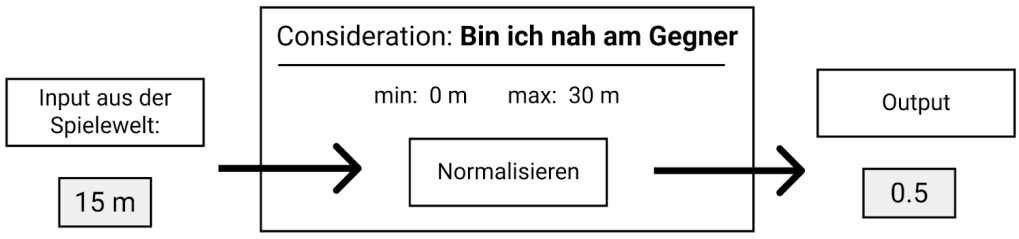


Abbildung 11: Bewertung einer Consideration

Wenn man annimmt, dass diese *Consideration* die Gefahr für einen Charakter symbolisiert, steigt die Gefahr, wenn die Entfernung abnimmt jeweils linear (s. Abbildung 12).

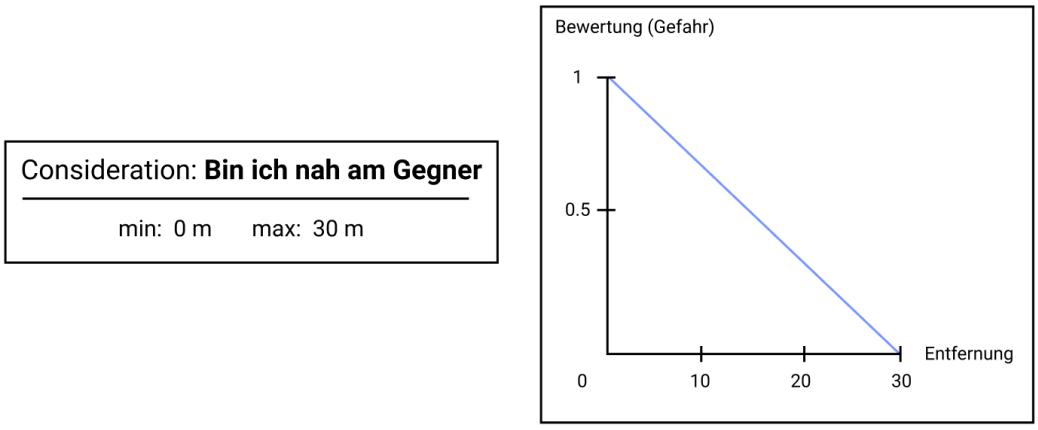


Abbildung 12: Verhältnis zwischen Gefahr und Distanz 1

Dieser lineare Zusammenhang ist jedoch nicht unbedingt damit vergleichbar, wie ein Mensch die Änderung der Gefahr wahrnehmen würde. Wenn sich einem Menschen beispielsweise eine Giftschlange nähern würde, würde er sich in derselben Gefahr fühlen, egal ob die Schlange 15 oder 13 Meter entfernt ist. Ob sich die Schlange jedoch 1 oder 3 Meter vom Menschen entfernt befindet, würde einen riesigen Unterschied für das Gefühl der Gefahr ausmachen.

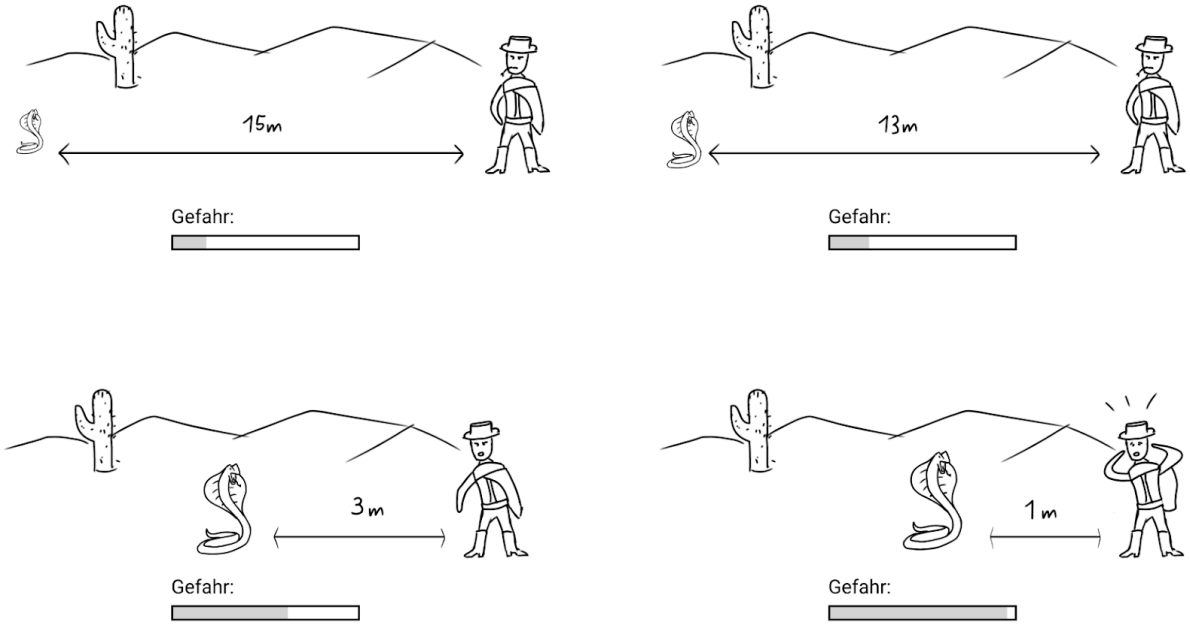


Abbildung 13: Verhältnis zwischen Gefahr und Distanz 2

In diesem Fall wäre ein exponentielles Verhältnis realistischer als ein lineares (s. Abbildung 14).

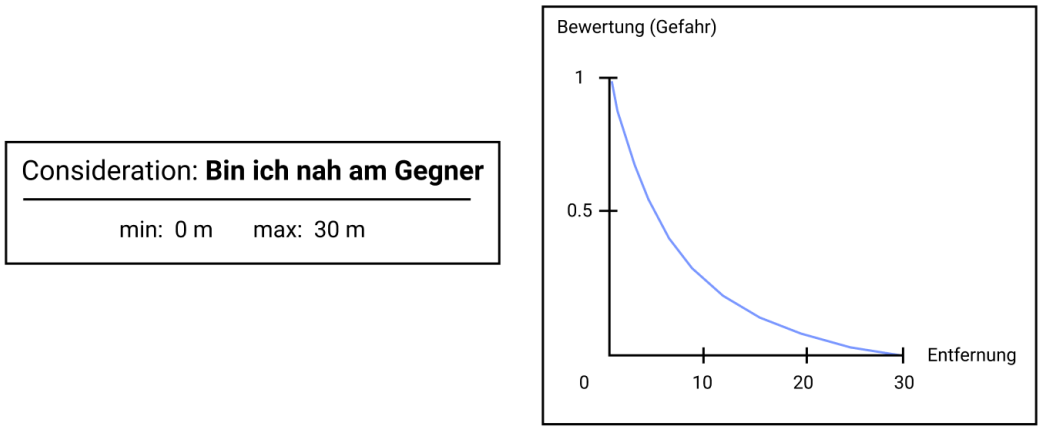


Abbildung 14: Verhältnis zwischen Gefahr und Distanz 3

Um verschiedene solcher Zusammenhänge simulieren zu können, schlägt Dave Mark in seinen Vorträgen deshalb vor, die numerische Bewertung in einer *Consideration* anschließend durch eine passende Kurve zu bearbeiten.

Die folgende *Consideration* wurde durch die oben abgebildete Exponentialfunktion angepasst.

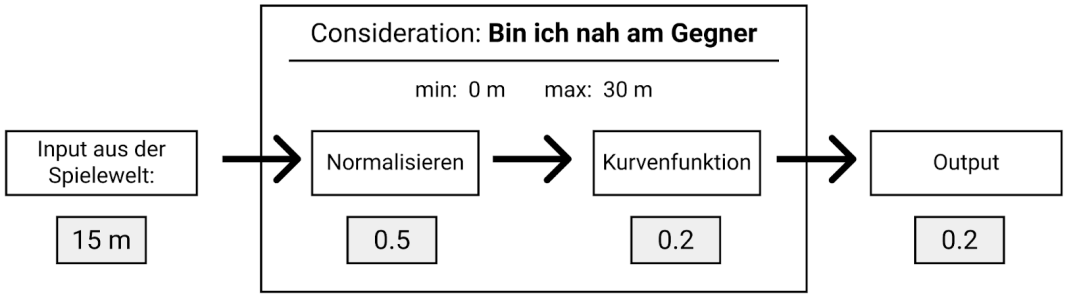


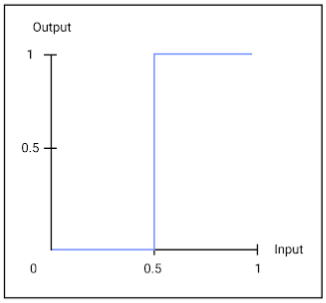
Abbildung 15: Bewertung einer Consideration mit Kurvenfunktion

Considerations: Kurven

Abhängig von dem gewünschten Verhältnis von Input zu der Bewertung kann man eine Vielzahl an verschiedenen Kurvenfunktionen einsetzen. Ich habe folgende 5 Kurven für die meine *Considerations* ausgewählt.

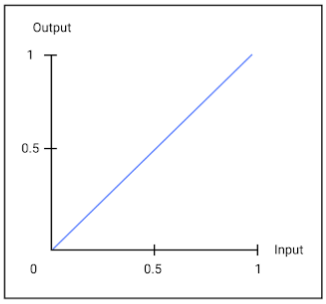
Die verschiedenen Parameter erlauben es, die Kurven für das gewünschte Verhältnis anzupassen.

Binary



Parameters:
[float] threshold
[float] verticalShift
[bool] inverse

Linear

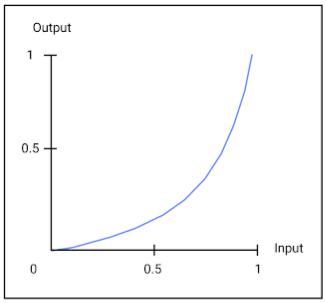


Parameters:
[float] slope
[float] shift

Formula:

output = slope * input + shift

Exponential

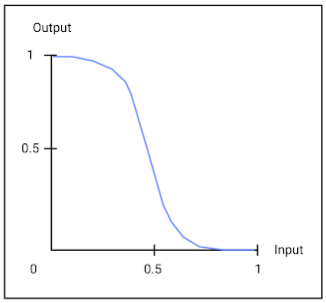


Parameters:
[float] slope
[float] exponent
[float] verticalShift
[float] horizontalShift

Formula:

$$output = slope * (input - horizontalShift)^{exponent} + verticalShift$$

Logistic

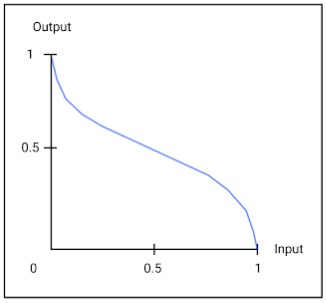


Parameters:
[float] slope
[float] verticalScalar
[float] verticalShift
[float] horizontalShift

Formula:

$$output = \frac{scalar}{(1 + e^{-slope * (input - (horizontalShift + 0.5))})} + verticalShift$$

Logit



Parameters:
[float] a
[float] b

Formula:

$$output = \frac{Log_e(input / (1-input) + a}{b}$$

Abbildung 16: Kurven für die Considerations

Die Formeln für die *Exponential*, *Logistic* und *Logit* Funktionen wurden aus den Folien von Dave Mark aus dem Vortrag **"Architecture Tricks: Managing Behaviors in Time, Space, and Depth"** aus der Game Developers Conference 2013 entnommen. Die Formeln für die *Logistic* und *Logit* Funktion wurde leicht von mir angepasst.

2.5.3 Zusammenfassung der Individual AI

Für das entwickelte Projekt wurden 2 ***Decision Maker*** genutzt. Der erste nennt sich *Positioning Layer* und trifft Entscheidungen darüber, wie sich der Charakter bewegen soll. Der zweite nennt sich *Interaction Layer* und befasst sich mit den Entscheidungen, welche Waffe ein Charakter wählen soll und wann er schießen oder nachladen soll. Die beiden *Decision Maker* können unabhängig voneinander Entscheidungen treffen.

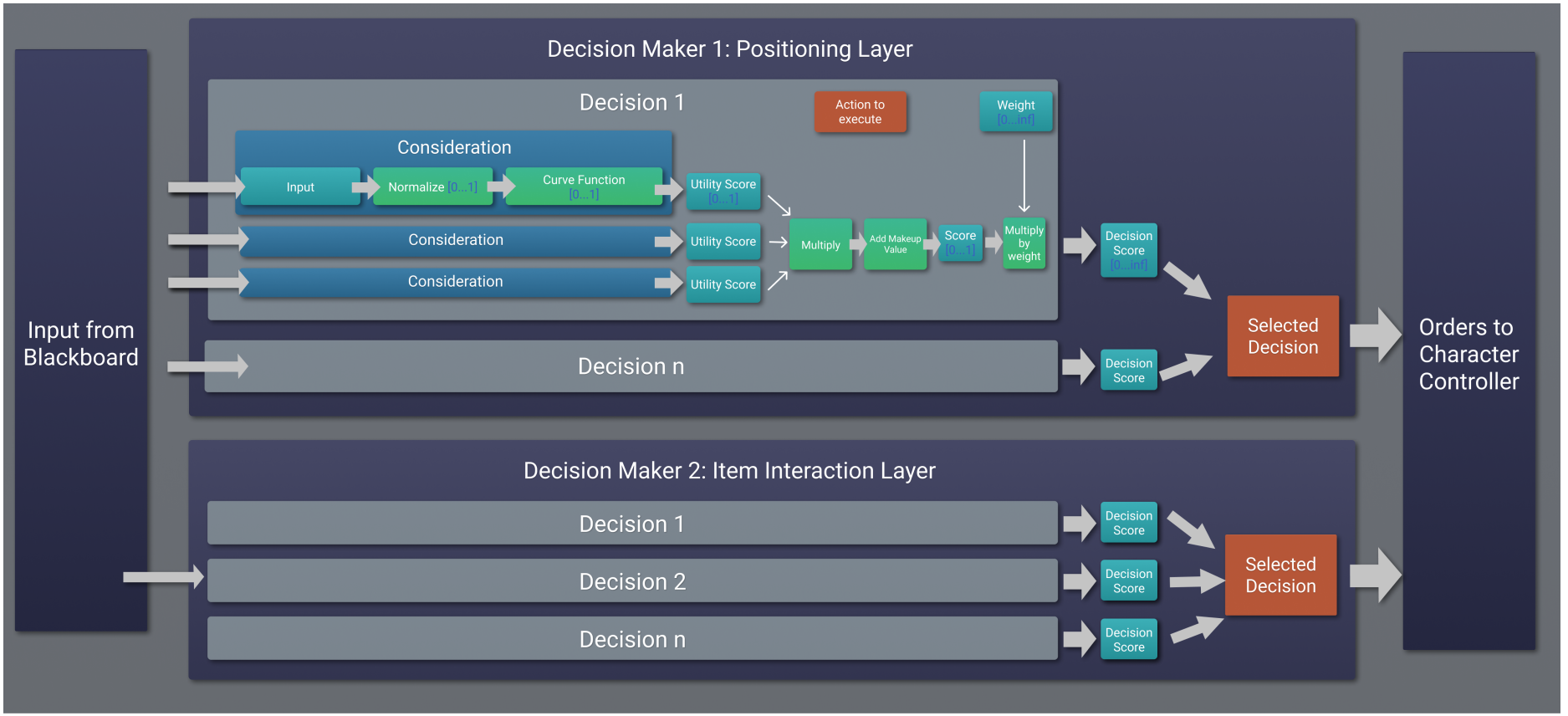


Abbildung 17: Zusammenfassung der Individual AI

2.6 Squad AI

Viele heutige Armeen basieren auf einer Form von Hierarchie. Jede kleine Gruppe von Soldaten hat einen Anführer, welcher ihnen Befehle erteilt. Dieser bekommt ebenfalls Befehle von einer höherrangigen Person. Die Simulation einer solchen Dynamik in einem Spiel könnte die Glaubwürdigkeit einer KI verbessern und neue spannende Gameplay-Möglichkeiten eröffnen. Ein Charakter könnte als Anführer einer Gruppe von Soldaten gekennzeichnet sein und diesen Soldaten Befehle geben, welche diese entgegennehmen und basierend auf deren Inhalt handeln.

Ich entschied mich jedoch für eine andere Art diese Dynamik zu implementieren, von der ich weniger Komplexität und mehr Kontrolle erwartet habe. Statt eines Charakters, welcher die Gruppe von Soldaten anführt, existiert eine allen Soldaten aus einer Gruppe übergeordnete separate Entität - die *Squad AI*. Dies ist eine KI, welche an keinen Soldaten gebunden ist und deren Position in der Spielewelt irrelevant ist. Während das Äquivalent zu der *Individual AI* in der realen Welt ein Gehirn eines Menschen wäre, könnte man sich die *Squad AI* als eine Art Geist vorstellen, der die Gehirne mehrerer Soldaten beeinflusst.

Die *Squad AI* ist optional, nicht jeder Agent muss von einer solchen KI beeinflusst werden. Sobald es narrativ Sinn ergibt oder das Gameplay dadurch verbessert wird, wird das Verhalten bestimmter Agenten durch die Squad-KI beeinflusst. Die *Squad AI* hat eine Anzahl an Befehlen, welche sie den Agenten erteilt und behält einen Überblick darüber, ob die Befehle erfolgreich ausgeführt wurden. Dies kann erfolgen, indem die befohlenen Agenten beispielsweise auf Position überprüft werden oder sich selbst nach erfolgreichem Ausführen der Befehle bei der *Squad AI* zurückmelden.

2.6.1 Erteilung der Befehle

Die Befehle werden folgendermaßen erteilt: Die leicht erweiterbare Natur des *IAUS* mit einer Liste an zur Verfügung stehenden *Decisions* erlaubt es, während des Spiels mit neuen *Decisions* erweitert zu werden. Die *Squad AI* erteilt Befehle, indem sie die Liste der *Decisions* der *Individual AI* mit vordefinierten, möglichst hoch gewichteten *Decisions* erweitert. Um Befehle zurückzunehmen, werden die jeweiligen Decisions der Liste wieder entnommen.

2.7 Debug - Visualisierung

Ein weiteres Ziel dieser Arbeit war es, eine Methode zu konzipieren, um Entscheidungen einer KI für den Entwickler nachvollziehbar zu visualisieren. Die Visualisierungen sollen es erlauben, die Entscheidungen einer KI einfacher nachzuvollziehen, um dadurch die Entwicklung zu vereinfachen und zu verkürzen. Dazu wurden folgende Methoden konzipiert:

2.7.1 Blackboard

Ein Soldat kann mit einem linken Mausklick ausgewählt werden. Sobald dieser selektiert wird, werden der KI aktuell zur Verfügung stehende Informationen in der UI angezeigt. Neben den Namen der wahrgenommenen Objekte, speichert das ***Blackboard*** die Information darüber, wie weit das Objekt entfernt ist und wann es das letzte Mal gesehen wurde.

Mit einem Klick auf bestimmte Informationen, bewegt sich die Kamera zu dem betroffenen Objekt in der Spielewelt hin. Das erleichtert dem Entwickler die Suche in der Szene.

Diese Informationen können, wenn die Option “Show in worldspace” eingeschaltet ist, auch in der Szene angezeigt werden, wie beim Soldaten Nummer 47988 (s. Abbildung 18).

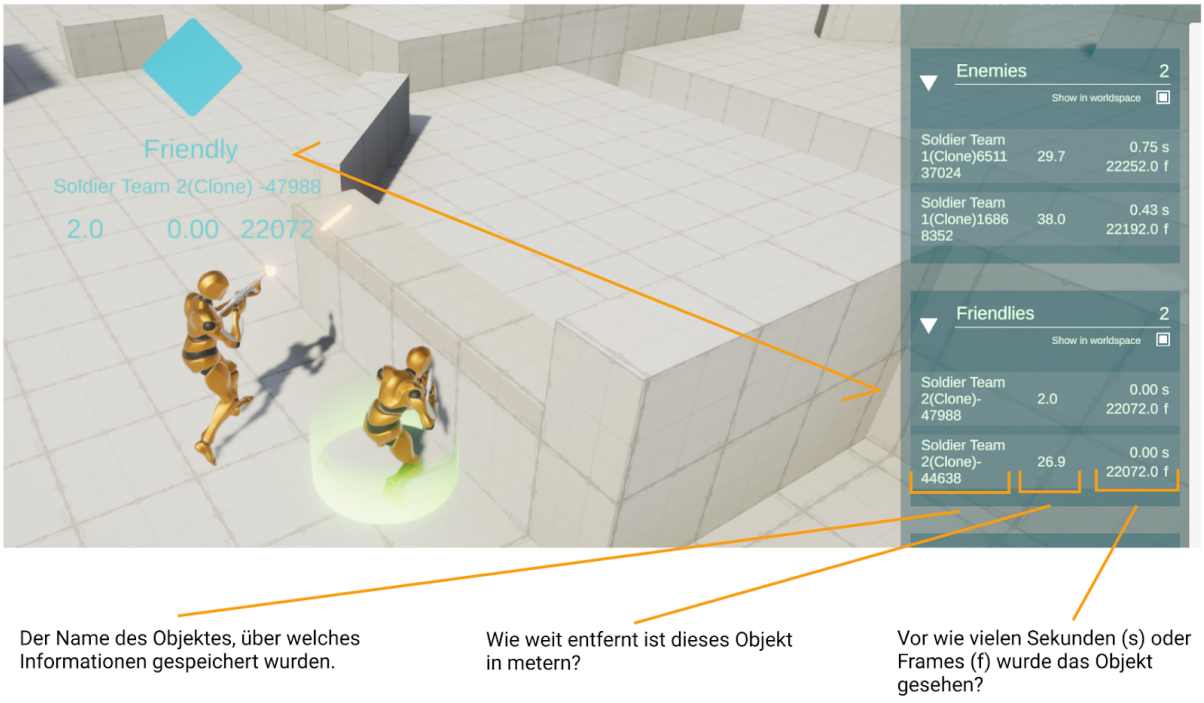


Abbildung 18: Blackboard Visualisierung

2.7.2 Current Decisions

Um schnell nachzuvollziehen, was eine KI aktuell macht, können die aktuell ausgewählten *Decisions* über dem Kopf eines Soldaten angezeigt werden. Für beide *Decision Layer* werden neben den Namen auch die Zeit angezeigt, an der diese Entscheidung getroffen wurde und welche Bewertung sie erhalten hat. Nebenbei werden hier auch die Lebenspunkte und der Stand des Magazins der benutzten Waffe angezeigt (s. Abbildung 19).

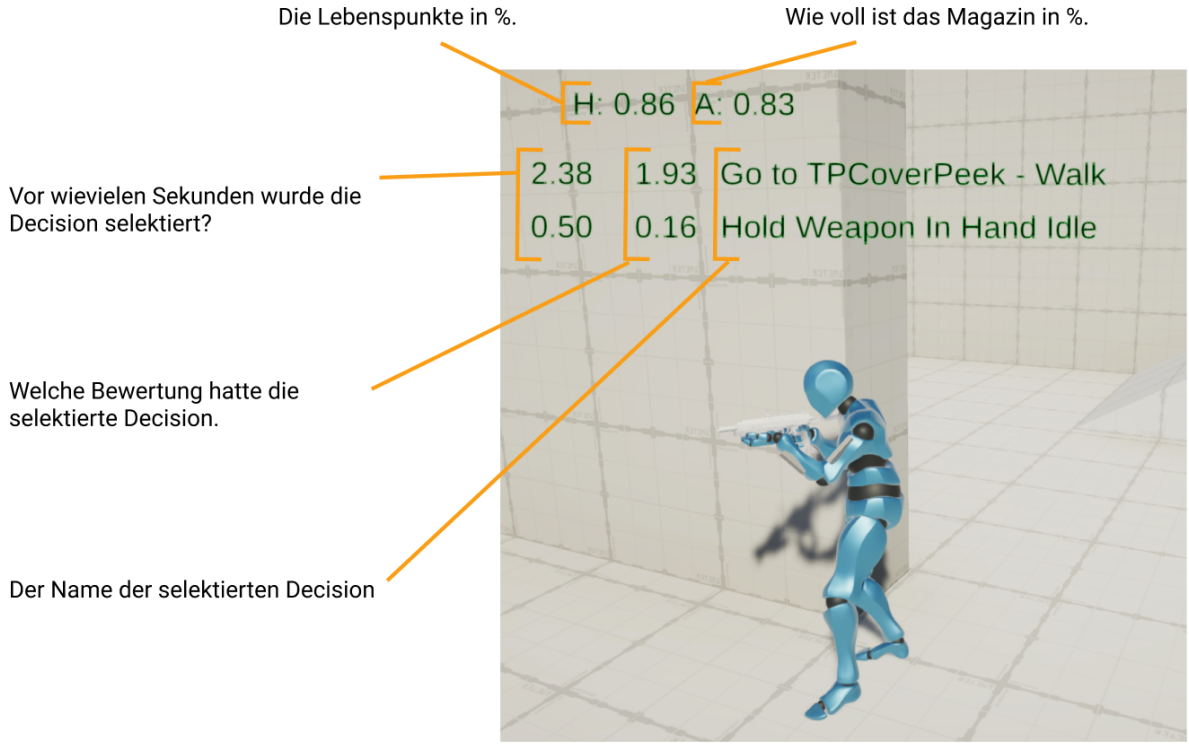


Abbildung 19: Simple Decision Visualisierung

2.7.3 Decision Making

Mehr Informationen über die Entscheidungen einer KI bietet das UI-Menü *Decision Making*. Auf der rechten Seite wird hier ein Verlauf der bisher selektierten *Decisions* gezeigt. Außerdem sind hier die *Considerations*/Gründe für die Auswahl aufgelistet. In dem linken UI-Panel können alle *Decisions* eingesehen werden, die der KI das letzte Mal zur Auswahl standen (s. Abbildung 20). Falls eine *Decision* ein Ziel hat, können die Namen der Ziele wie Buttons angeklickt werden, damit die Kamera sich zu diesen Zielen bewegt. Dies vereinfacht die Suche in der Szene.

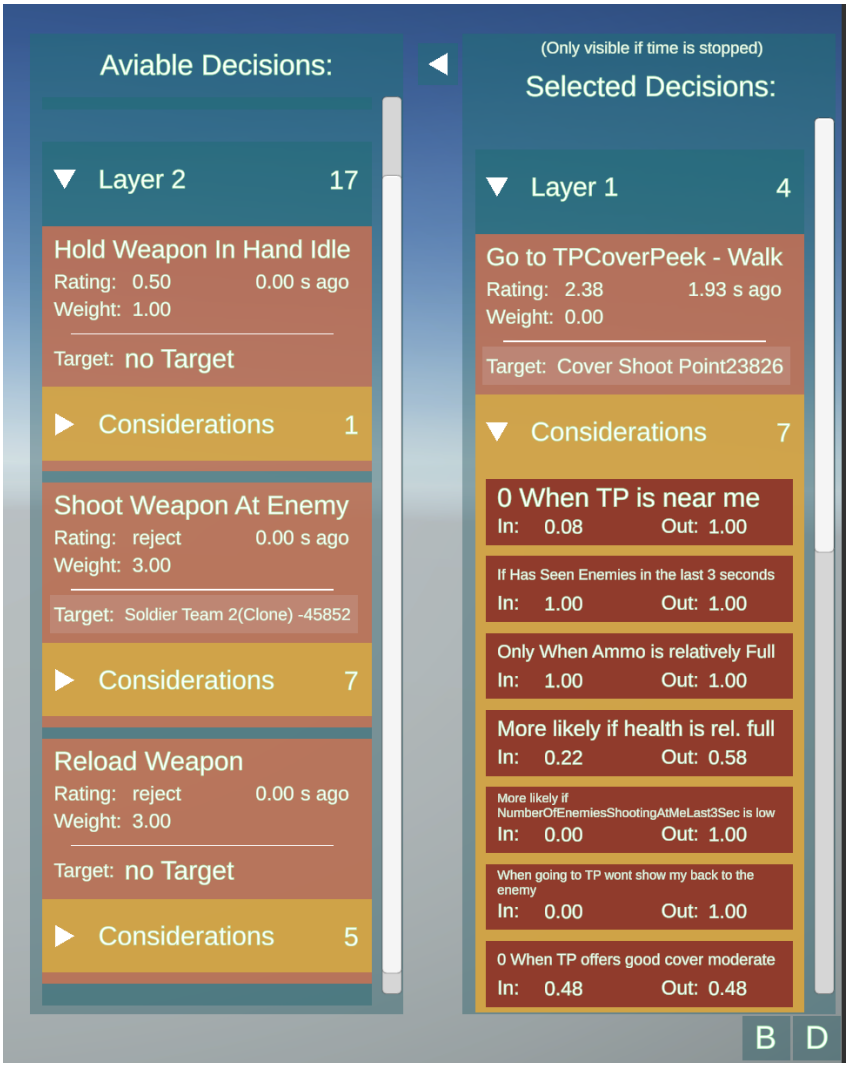


Abbildung 20: Erweiterte Decision Visualisierung

3 Implementierung

Im Folgenden werden die interessantesten Herausforderungen vorgestellt, die bei der Implementierung des konzipierten Systems aufgekommen sind. Aufgrund des großen Umfangs des Projektes werden Teile der Implementierung ausgelassen.

3.1 Begriffserklärung

3.1.1 GameObject

Dies sind Objekte in einer Szenenhierarchie. Jedes GameObject besitzt eine Position, Rotation und Scale im Koordinatensystem der Szene oder des jeweiligen parent-GameObjects und einen Namen.

3.1.2 MonoBehaviour / Component / Script

Ein GameObject alleine ist ein unsichtbares Objekt mit Namen und Position. Um die Funktionsweise eines GameObjects zu erweitern, werden diesem Components (auch *Scripts* oder MonoBehaviours genannt) hinzugefügt. Dies sind Klassen in C#, welche von der Klasse Monobehaviour erben. Die Unity Engine führt bestimmte Methoden in den MonoBehaviours während des Spiels aus. Beispielsweise wird jeden Frame die Methode `Update()` ausgeführt, sobald diese vorhanden ist. Beispielsweise gibt es Renderer-Components, die Meshes visualisieren, Collider- und Rigidbody-Components, die die Interaktion mit der Physik-Engine erlauben oder PlayerController-Components, die ein *GameObject* basierend auf dem Input vom Spieler bewegen.

3.1.3 Prefab

Prefabs speichern GameObject zusammen mit angehängten Components für die spätere Nutzung in einer .prefab Datei in der Ordnerstruktur des Projektes. Diese Prefab Datei stellt eine Art Vorlage dar, basierend auf welcher Instanzen dieses Prefabs in einer Szene platziert werden können. Beispielsweise werden alle Soldaten in diesem Projekt aus dem Prefab “Soldier” erstellt.

3.1.4 ScriptableObjects

ScriptableObjects werden primär als Datencontainer genutzt. Sie können aber auch Methoden besitzen, welche von anderen Klassen ausgeführt werden. Im Unterschied zu MonoBehaviours, weden ScriptableObjects nicht zu einem GameObject als Component hinzugefügt. Stattdessen werden sie als Dateien in der Ordnerstruktur des Projektes gespeichert. ScriptableObjects können dazu genutzt werden, den benötigten Speicher eines Projektes zu reduzieren oder auch um eine Klassenstruktur übersichtlicher zu gestalten. Eine weitere Einsatzmöglichkeit ist das Nutzen von ScriptableObjects als “Enum mit Methoden und Feldern”. Auf diese Art wird ein Großteil der ScriptableObjects in diesem Projekt benutzt.

3.1.5 EditMode & PlayMode

Die meiste Zeit verbringt ein Entwickler in Unity im EditMode. In diesem Modus können GameObjects in die Szene platziert und mit Components versehen werden. Sobald der Play-Button gedrückt wird, wechselt die Unity Engine in den PlayMode, bis er wieder mit dem Play-Button verlassen wird. Der PlayMode simuliert das Gameplay in einem dafür vorgesehen Fenster. Die Methoden der meisten MonoBehaviours werden ausschließlich im PlayMode ausgeführt.

3.2 Testszenario

Um testen zu können, wie gut die Teile des Projektes implementiert wurden und wie sie funktionieren, wurden in Unity zwei Szenen erstellt und mit einem Szenario ausgestattet. Die Szenen wurden per Hand mit Hindernissen und *Tactical Points* befüllt.

Das Testszenario zeigt Soldaten in 2 verschiedenen Teams, die sich gegenseitig bekämpfen. Beide Szenen haben einen **`ScenarioManager`**. Dieser spawnt in einem zufälligen Intervall zwischen 1 und 3 Sekunden für jedes Team einen Soldaten und gibt diesem den Befehl zu dem gegnerischen Spawn zu laufen. Durch die UI kann der Spieler einige Werte des `SzenarioManagers` anpassen, wie die maximale Anzahl an Soldaten oder ob der `SzenarioManager` das Spawnen stoppen soll.

Nach dem Spawnen sorgt die *Individual AI* der jeweiligen Soldaten dafür, dass der vom `ScenarioManager` gegebene Befehl auch ausgeführt wird und entscheidet, wie gegen auf dem Weg dorthin getroffene Gegner vorgegangen wird.

Abbildung 21: Testszenario

3.3 Character Controller

Für die Implementierung des *Character Controllers* wurde folgende Script-Architektur verwendet:

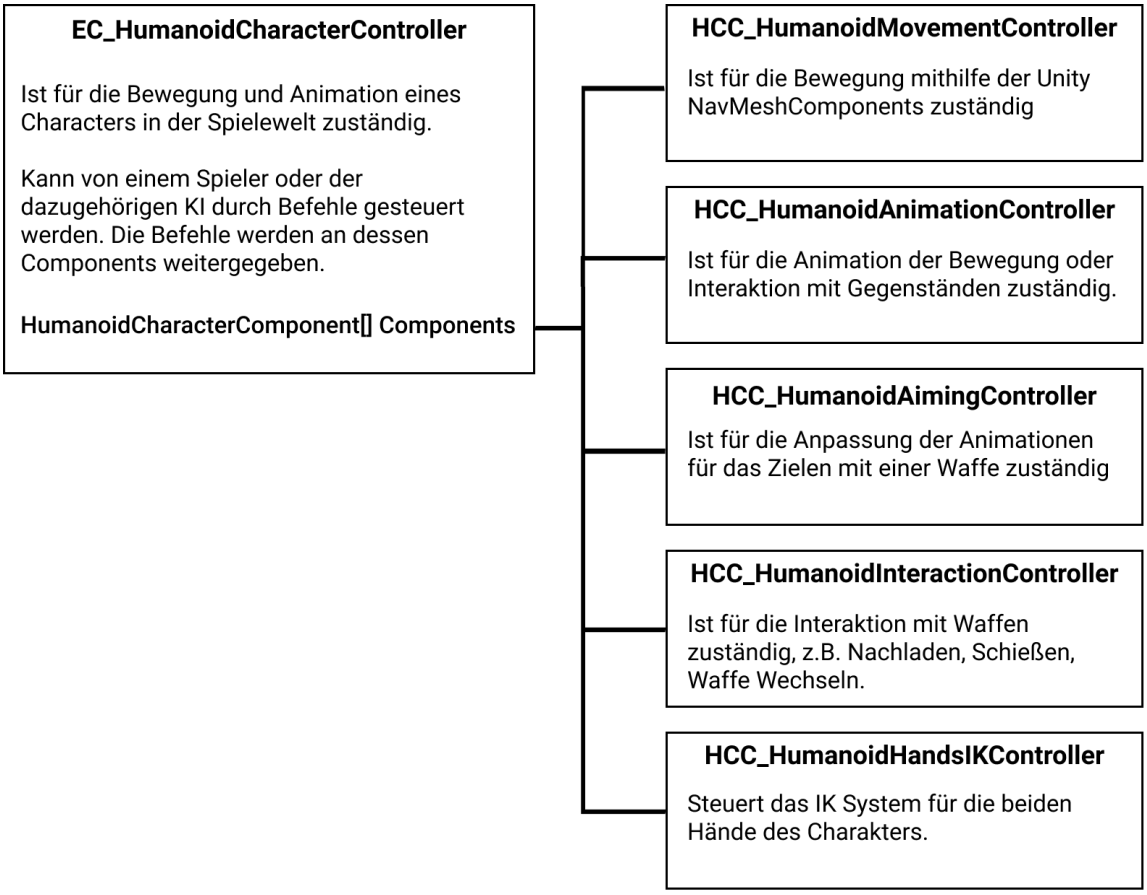


Abbildung 22: Implementierung des Character Controllers

Der **`EC_HumanoidCharacterController`** wird durch 5 Sub-Controller erweitert, welche verschiedene Teile eines Charakters kontrollieren.

Für die Bewegung des Charakters werden in der Klasse **`HCC_HumanoidMovementController`** die **`NavMeshComponents`** von Unity benutzt, welche eine Erweiterung des normalerweise in der Unity-Engine vorhandenen Navigationssystem sind.

Die Animationen wurden Mithilfe des Unity-Animators implementiert, welcher durch das Script **`HCC_HumanoidAnimationController`** gesteuert wird. Für dieses Projekt wurden kostenlos verfügbare Animationen aus der Webseite **Mixamo** verwendet. Es wurden “In-Place” Animationen anstatt von “Root-Motion” Animationen benutzt, um die Implementierung zu vereinfachen.

Diese werden durch prozedurale Animationen für das Zielen der Waffe und des ganzen Oberkörpers in dem Script **`HCC_HumanoidAimingController`** erweitert.

Außerdem kümmert sich das Script **`HCC_HumanoidHandsIKController`** um die Anpassung der Hände an die Waffe mithilfe von Inverse Kinematics.

Dieser *Character Controller* ist im entwickelten Projekt im Prefab “Soldier” vorzufinden.

3.4 Abstract World Representation

Die *Tactical Points* sind Prefabs und können vom Entwickler an beliebigen Orten in der Szene per Hand platziert werden. Die Bewertungen der *Tactical Points* wird mithilfe von Raycasts automatisch berechnet. Diese Automatisierung soll wertvolle Zeit sparen, die eine manuelle Eingabe der Bewertungen mit sich ziehen würde.

3.4.1 Automatisierte Bewertung der Tactical Points

Die Bewertungen der *Tactical Points* findet im EditMode statt. Ähnlich wie das *Lightmapping* kann sie per Knopfdruck gestartet werden und wird per Szene gespeichert.

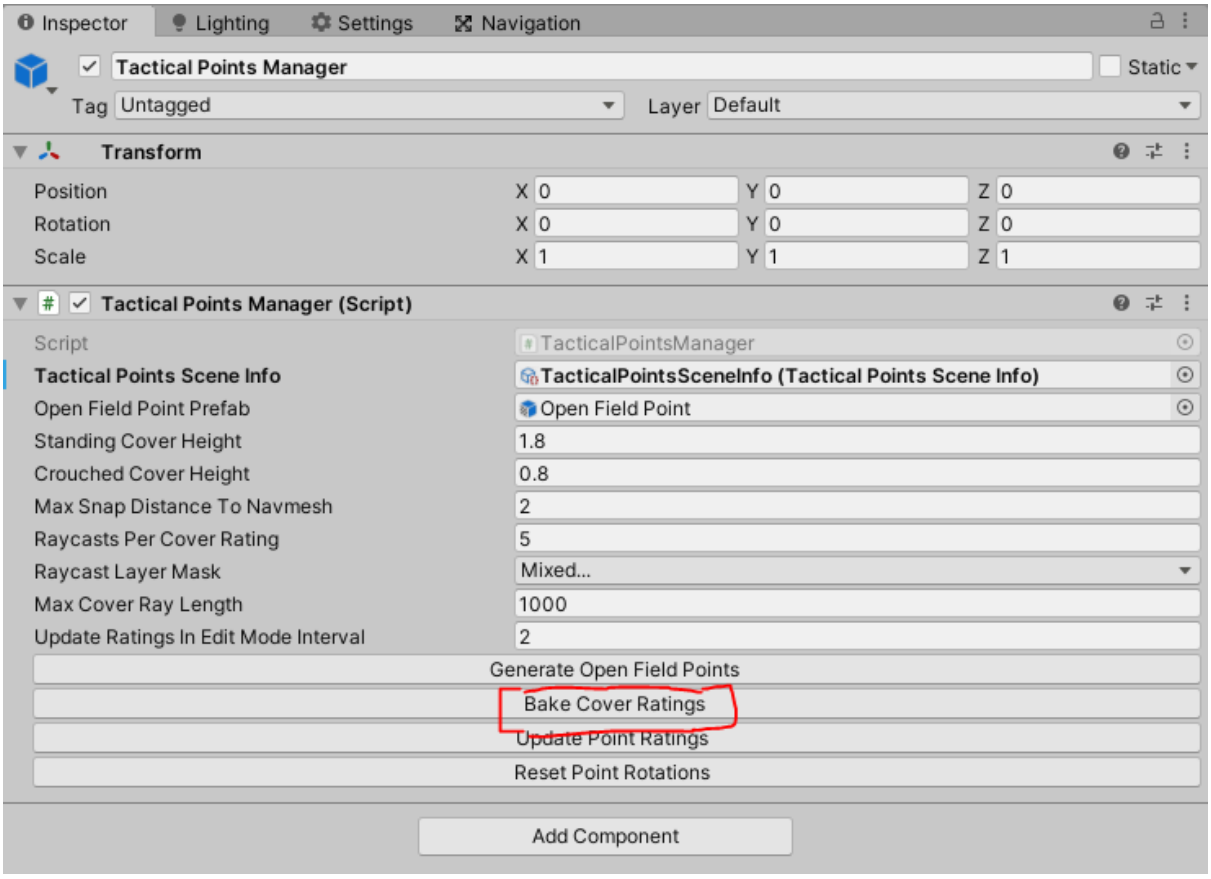


Abbildung 23: Inspector des TacticalPointsManagers

Der **`TacticalPointsManager`** ist ein Singleton und hat eine Referenz zu jedem *Tactical Point* in der Szene. Außerdem hat der Manager Parameter, auf denen die Bewertung der *Tactical Points* basiert. Durch das Drücken des Buttons wird durch alle *Tactical Points* in der Szene iteriert und diese werden mithilfe von Raycasts des Physik-Systems von Unity bewertet.

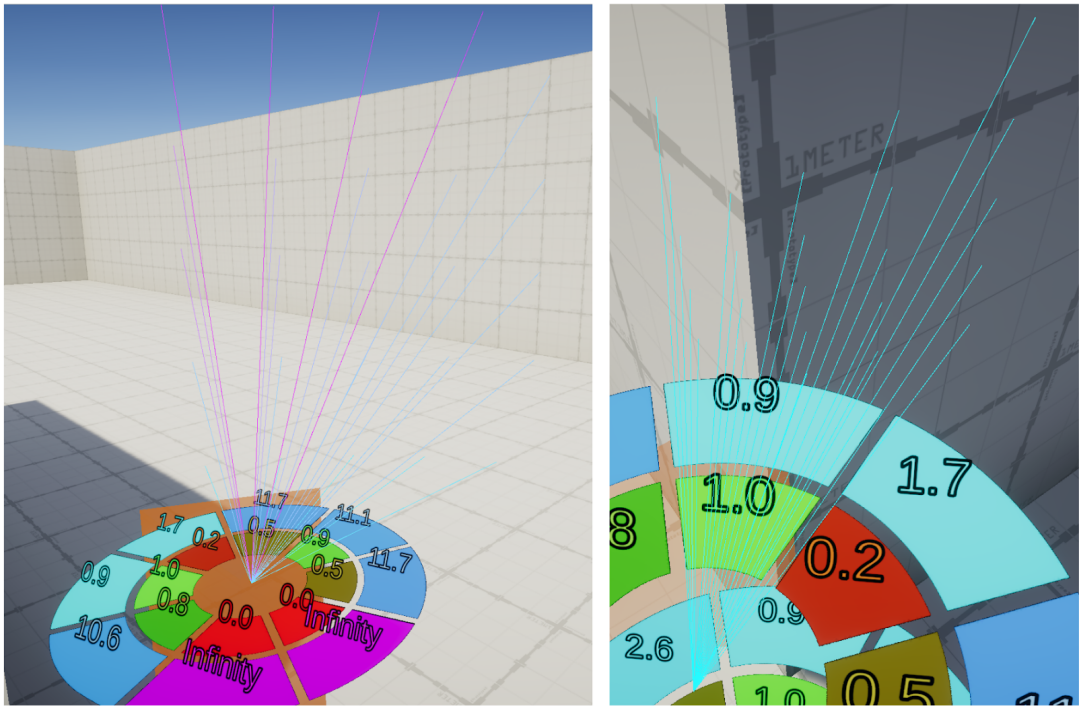


Abbildung 24: Visualisierung der Bewertung von TPoints

In jede der 8 Richtungen werden in gleichmäßigen Abständen zueinander eine bestimmte Anzahl an Raycasts geschossen (s. Abbildung 24). Für jeden dieser Raycasts wird die Information gespeichert, ob mit etwas kollidiert wurde und in welcher Entfernung dies passiert ist.

Die Bewertung kann basierend auf diesen Daten auf verschiedene Weise berechnet werden. Wichtig für die Bewertung ist, dass die von der KI schnell und performant gelesen werden kann, denn dies muss im Spiel passieren. Die Geschwindigkeit des Algorithmus für die Berechnung der Bewertung spielt keine Rolle, da dies nur im EditMode passieren wird.

Distance Rating

Als Erstes wird das ***Distance Rating*** berechnet. Dies ist der Modalwert der Distanz aller Raycasts. Anders bezeichnet ist dies die Zahl, die als Distanz am öftesten vorkommt. Da die Distanzen Floats sind, ist es sehr unwahrscheinlich, dass zwei gleiche Zahlen existieren. Um trotzdem einen verlässlichen Modalwert zu bekommen, werden sich ähnelnde Distanzen gruppiert. Jede Gruppe erlaubt nur eine Differenz von maximal 1 unter ihren Mitgliedern. Der Mittelwert der Distanzen aus der größten Gruppe ist das Distance Rating.

Die Distanzen der Raycasts (sortiert):

[1.3 | 1.32 | 1.73 | 1.78 | 2.1 | 2.4 | 2.5 | 3.72 | 3.8 | 6.23 | 7.2 | Inf. | Inf.]

1. Gruppieren:

Gruppe 1 [5] : [1.3 | 1.32 | 1.73 | 1.78 | 2.1]

Gruppe 2 [2] : [2.4 | 2.5]

Gruppe 3 [2] : [3.72 | 3.8]

Gruppe 4 [2] : [6.23 | 7.2]

Gruppe 5 [2] : [Inf. | Inf.]

2. Mittelwert aus der größten Gruppe bilden:

-> Gruppe 1 [5]

$$\textbf{Distance Rating} = \frac{1.3 + 1.32 + 1.73 + 1.78 + 2.1}{5} = 1.646$$

Abbildung 25: Berechnung des Distance Ratings

Quality Rating

Das ***Quality Rating*** entspricht dem prozentualen Anteil an Raycasts, welche kürzer waren als der längste Raycast aus der bei dem *Distance Rating* ausgewählten Gruppe.

Die beim DistanceRating ausgewählte Gruppe:

Gruppe 1 [5] : [1.3 | 1.32 | 1.73 | 1.78 | 2.1]

Folgende Distanzen waren <= 2.1 :

[1.3 | 1.32 | 1.73 | 1.78 | 2.1]

-> Dies sind 5 von 13 Distancen

Quality Rating = 5/13 = ~0.38

Abbildung 26: Berechnung des Quality Ratings

Die beiden Algorithmen für die Bewertung befinden sich in der Klasse **`TacticalPoint`** in der Methode `BakeCoverRatings()`

3.4.2 Speicherung der Bewertung

Die Bewertung der *Tactical Points* wird nicht in dem GameObject selbst gespeichert. Stattdessen existiert ein separates Datenobjekt, welches hierfür verwendet wird. Dies ist ein ScriptableObject, welches sich im Ordner der jeweiligen Szene befindet. Das Objekt nennt sich “Tactical Points Scene Info” und speichert die Ratings aller Points aus einer Szene. Diese werden während der Bewertung im EditMode darin gespeichert und zu Beginn des PlayModes von den **`TacticalPointManager`** in der Methode `UpdatePointRatings` ausgelesen und an die *Tactical Points* in der Szene geschickt.

Die Vorteile einer solchen Speicherung der Daten in einem separaten Objekt, statt direkt in der Szene, waren einerseits die durch Zentralisierung bedingte verbesserte Leserlichkeit des Codes und ein scheinbar geringerer Speicherverbrauch.

3.5 Sensing

3.5.1 Informationen über die Umgebung gewinnen

Um Informationen über die Umgebung zu erhalten, scannt die KI die Umgebung mithilfe des Physik-Systems von Unity in regelmäßigen zeitlichen Abständen. Dafür ist die Klasse **`AIC_HumanSensing`** zuständig.

Um die Umgebung zu scannen, wird die Methode `Physics.OverlapSphere()` benutzt, welche ein Array an Collidern in einem bestimmten Radius zu dem Soldaten zurückgibt. Diese Methode wurde gewählt, da das Physik-System diese Abfrage sehr performant ausführt.

Collider sind Components, die GameObjects angehängt werden können. Alle Objekte, die von der KI wahrgenommen werden sollen, sind mit einem solchen Collider ausgestattet. Normalerweise werden Collider verwendet, um Kollisionen zu simulieren. Um Verwirrung und Bugs durch die Benutzung von Collidern für zwei verschiedene Zwecke zu verhindern, erlaubt Unity verschiedene Collision-Layer einzurichten. Collider auf zwei verschiedenen Layern können sich komplett ignorieren, sobald dies in der Collision-Matrix eingestellt wurde (s. Abbildung 27). Die Collision-Layer für das Sensing kollidieren mit nichts, sie werden vom Physik-System nur bei Abfragen wie `OverlapSphere()` beachtet. Außerdem erhöhen gut eingestellte Collision-Layer die Performance anderer Physik-Abfragen.

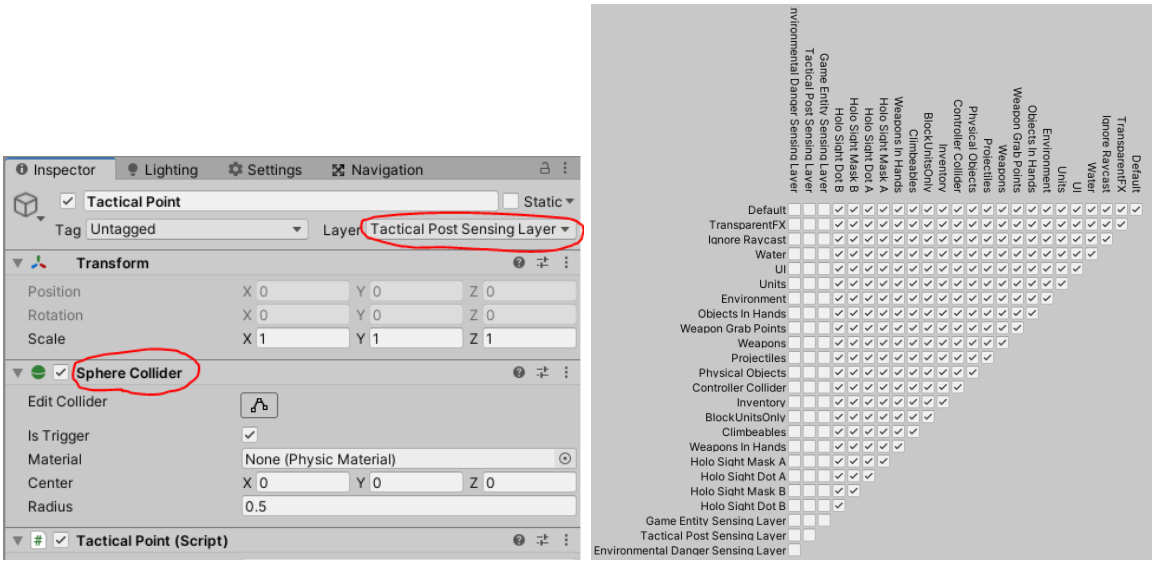


Abbildung 27: Tactical Point Collider & Collision-Matrix

Erkennung von anderen Soldaten

Beim scannen nach anderen Soldaten wird nur der Collision-Layer “Game Entity Sensing Layer” abgefragt. Dadurch symbolisiert jeder der durch die Methode `Physics.OverlapSphere()` zurückgegebenen Collider, einen Soldaten. Jedoch sind nicht alle dieser Soldaten für den anfragenden Agenten sichtbar. Manche können sich hinter ihm befinden oder sich hinter einer Wand verstecken. Diese versteckten Soldaten befinden sich zwar im Radius und werden von der Methode `Physics.OverlapSphere()` auch zurückgegeben, sollten aber nicht gesehen werden. Deshalb müssen die von `Physics.OverlapSphere()` zurückgegebenen Collider gefiltert werden.

Als Erstes werden alle Collider in den Local Space des Kopfes des Soldaten, welcher seine Umgebung scannt, transformiert. Nun werden alle Collider mit einer lokalen z-Position von <0 aussortiert. Diese befinden sich hinter dem Soldaten, sollten also nicht sichtbar sein. Für alle restlichen Soldaten wird ein Line of Sight Test durchgeführt. Ein Raycast wird vom Kopf des Soldaten zu einem Punkt auf dem Körper eines anderen Soldaten geschossen. Falls der Raycast dort ankommt, ohne mit anderen Objekten zu kollidieren, war der Line of Sight Test erfolgreich und der Soldat wird erst jetzt “gesehen”.

Um das Hören eines Menschen zu simulieren und um mögliche Fehler des vorgestellten Systems zu vertuschen, werden alle Soldaten in einem kleinen `hearingRadius` sofort erkannt, ohne einen Line of Sight Test.

Erkennung von Tactical Points

Tactical Points werden von der KI wahrgenommen, sobald sie in einem bestimmten Radius sind. Alle Collider, die von der Methode `Physics.OverlapSphere()` zurückgegeben werden, werden ungefiltert übernommen.

Erkennung von Environmental Dangers

Environmental Dangers sind ein Sammelbegriff für Objekte in der Welt, welche gefährlich, jedoch keine Gegner sind. In der Praxis können dies Feuer oder Gift auf dem Boden sein. Aktuell hat das Projekt nur einen *Environmental Danger* - Granaten. Die Granaten werden von der KI erkannt, sobald sich diese in einem bestimmten Radius befinden. Alle Collider, die von der Methode `Physics.OverlapSphere()` zurückgegeben werden, werden ungefiltert übernommen.

3.5.2 Informationen über die Umgebung speichern

Die Klasse `AIC_HumanSensing` schickt die von der Umgebung gewonnen Informationen an die Klasse **`AIController_Blackboard`**. Das ***Blackboard*** speichert die Informationen in public Arrays, welche dann von der KI gelesen werden können. Oft fragt die KI nach dem Gegner, der sich am nähesten befindet. Damit diese Distanz-Abfrage nicht unnötig viel Rechenleistung verbraucht, sind die Arrays im *Blackboard* nach der Distanz zur KI sortiert.

Informationen über andere Soldaten werden in speziellen Container-Objekten namens **`SensedEntityInfo`** gespeichert. Neben Informationen über die Art des Soldaten und seiner

Position, speichert dieser Container auch Informationen darüber, wann dieser Soldat gesichtet wurde. Dadurch kann die KI beispielsweise Gegner, die sie das letzte Mal vor 10 Sekunden gesehen hat, als weniger wichtig einstufen, als einen Gegner der vor einer Sekunde gesichtet wurde. Neue Informationen über einen Soldaten überschrieben die älteren, sobald diese schon im *Blackboard* vorhanden sind.

3.6 Individual AI

3.6.1 AI Controller

Der **`AIController`** ist der für die KI zuständige Klasse. **`AIController_HumanoidSoldier`** erbt von dieser Klasse und wird zu einem der GameObjects in dem "Soldier" Prefab hinzugefügt. Der `AIController` der Soldaten beinhaltet in diesem Projekt zwei ***Decision Maker***, welche wiederum eine Sammlung an ***Decisions*** beinhalten.

Der **`DecisionMaker`** sowie die **`Decision`** sind Klassen, die nicht von MonoBehaviour erben. Dadurch können sie nicht an GameObjects angehängt werden. Sie können jedoch als Felder in dem `AIController` festgelegt werden, wodurch sie vom Entwickler im Inspektor des `AIControllers` festgelegt werden können.

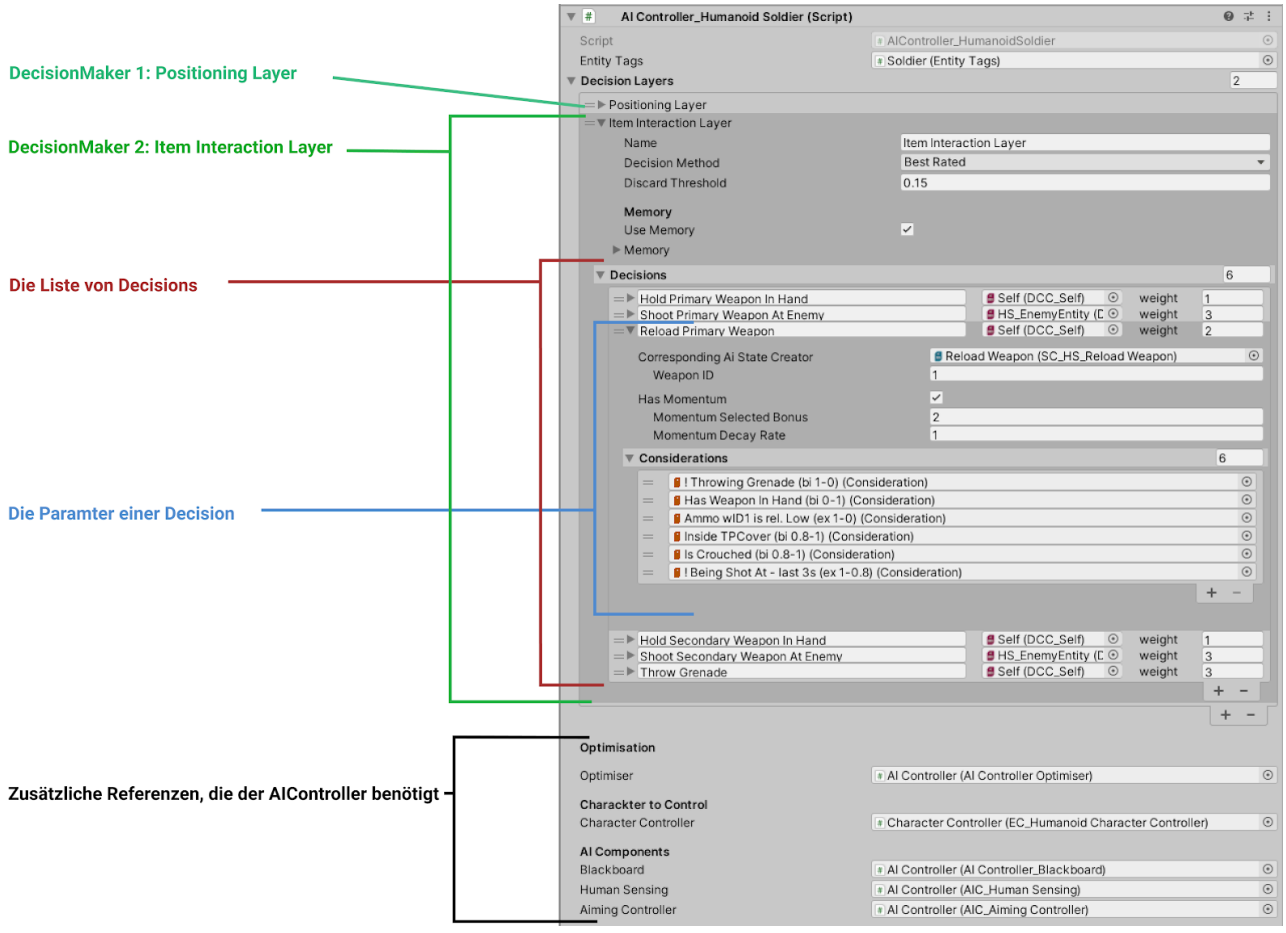


Abbildung 28: Inspector des AIControllers

3.6.2 Decide() Methode

In einem festgelegten zeitlichen Intervall ruft der `AIController` in jedem der `DecisionMaker` die `Decide()` Methode auf. Diese Methode ist das Kernstück des "Infinity Axis Utility System". Hier werden alle `Decisions` im `DecisionMaker` bewertet und die Aktion der am besten bewerteten `Decision` ausgeführt.

Codebeispiel 1: Die Aufrufen der Decide() Methode im AIController:

```
protected virtual void UpdateDecisionMakers()
{
   for (int i = 0; i < decisionLayers.Length; i++)
   {
      decisionLayers[i].Decide();
   }
}
```

Codebeispiel 2: Die Decide Methode im DecisionMaker:

```
public void Decide()
{
   // Score all decisions, select the best one and create a new state if this decision
     is different from the previous decision.
   float currentRating = 0;
   float bestRatingSoFar = 0;
   DecisionContext bestRatedDecisionContext = new DecisionContext();

   for (int i = 0; i < decisions.Count; i++)
   {
      DecisionContext[] decisionContexesToAdd
      = decisions[i].GetRatedDecisionContexts(aiController, discardThreshold);

      for (int j = 0; j < decisionContexesToAdd.Length; j++)
      {
          currentRating = decisionContexesToAdd[j].rating;

          if (currentRating > bestRatingSoFar)
          {
             bestRatingSoFar = currentRating;
             //the Decision context needs to be copied, as it is a reference to an
               object from a pool which can change during runtime.
             bestRatedDecisionContext.SetUpContext(decisionContexesToAdd[j]);
          }
       }
    }

    if(bestRatedDecisionContext.IsContextValid())
    {
       StartExecutingDecision(bestRatedDecisionContext);
    }
}
```

3.6.3 Ausführen einer Decision

Um eine Aktion auszuführen, wird in der Methode `StartExecutingDecision()` des `DecisionMaker` ein zu der selektierten `Decision` passender `AIState` erstellt. Der `DecisionMaker` behält eine Referenz zu diesem `AIState` und führt diesen, unabhängig von der `Decide()` Methode, jeden Frame aus.

Wenn sich die aktuelle `Decision` nicht von der vorher gewählten `Decision` unterscheidet, ist das Erstellen eines neuen `AIState`-Objektes nicht nötig.

Codebeispiel 3: Ausführen einer Decision

```
public void StartExecutingDecision(DecisionContext decisionContext)
{
   if (!decisionContext.ContextIsTheSameAs(lastSelectedDecisionContextMemory))
   {
      if (currentState != null)
      {
         currentState.OnStateExit();
      }

      currentState = decisionContext.decision.CreateState(aiController,decisionContext);
      currentState.OnStateEnter();

      lastSelectedDecisionContextMemory
      = new Memory.DecisionContextMemory(decisionContext, 0);
   }
}
```

3.6.4 AI State

`AIStates` geben dem *Character Controller* Befehle. Beispielsweise sorgt der State "ShootWeaponAtEnemy" dafür, dass ein Soldat einen anderen mit einer Waffe anschießt. Das Grundprinzip der KI ist es, zwischen diesen verschiedenen States mithilfe der *Decisions* zu wechseln.

Ein State hat verschiedene Hooks, welche ihm erlauben die Befehle und Änderung, die er vornimmt zu managen und vollzogene Änderungen bei Bedarf wieder zum ursprünglichen Zustand zurückzubringen.

Codebeispiel 4: AI State

```
public abstract class AIStateCreator : ScriptableObject
{
   [HideInInspector]
   public AIStateCreatorInputParams.InputParamsType[] inputParamsType;

   // Creates and returns an AIState object.
   public abstract AIState CreateState
   (
      AIController aiController, DecisionContext context,
      AIStateCreatorInputParams inputParams
   );
}
```

```
public abstract class AIState
{
   // Deriving classes will all implement their own Constructor with needed info as
      parameters.
   //public abstract void SetUpState(AIController aiController, DecisionContext
      context);

      public abstract void OnStateEnter();

      public abstract void OnStateExit();

      public abstract EntityActionTag[] GetActionTagsToAddOnStateEnter();

      public abstract EntityActionTag[] GetActionTagsToRemoveOnStateExit();

      public abstract void UpdateState();

      public abstract bool ShouldStateBeAborted();
}
```

States werden durch einen `AIStateCreator` erstellt. Dies ist ein ScriptableObject, welches wie ein Enum im Inspektor der `Decision` ausgewählt werden kann.

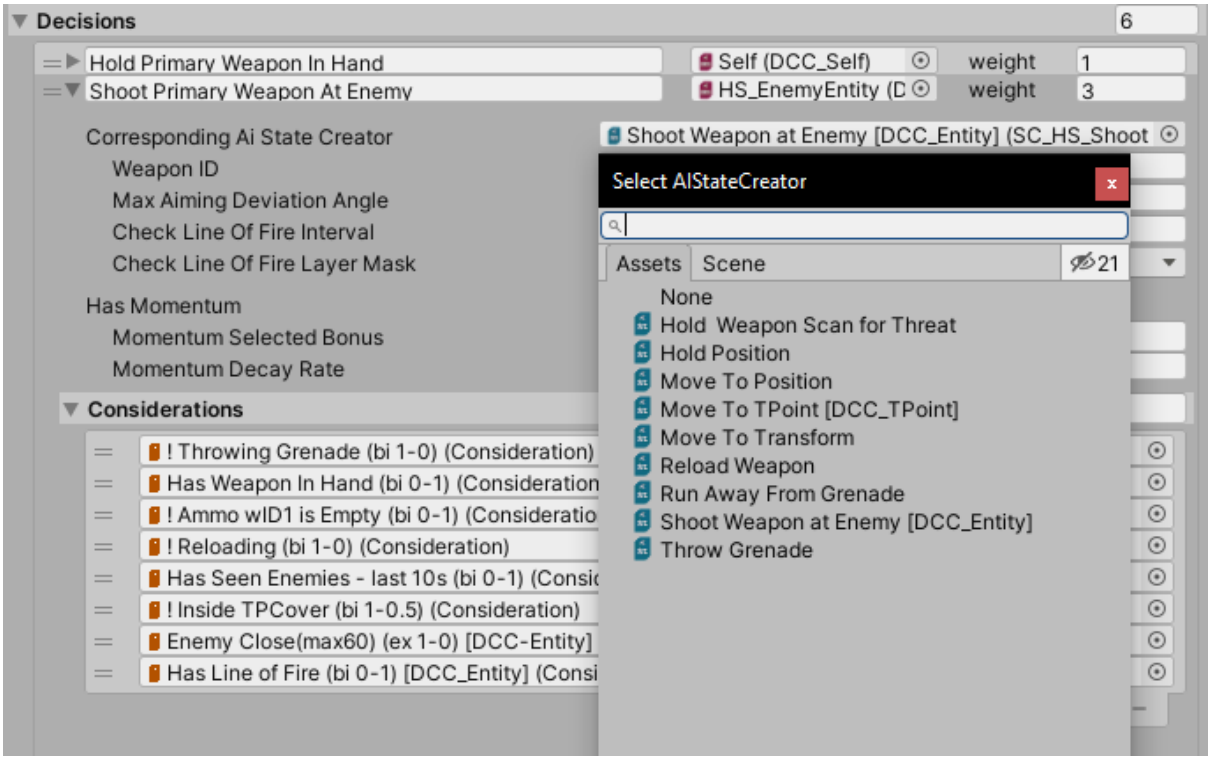


Abbildung 29: Auswahl eines AIStates im Inspector der Decision

3.6.5 Decision Context

DecisionContexts sind das Interface zwischen Decisions und dem DecisionMaker. Beispielsweise kann sich die Decision "ShootAtEnemy" auf mehrere Ziele beziehen. Für jedes dieser Ziele wird in der Decision Klasse ein DecisionContext Objekt vorbereitet. Dadurch kann eine Decision mehrere DecisionContext Objekte zurückgeben, welche alle bezogen auf andere Ziele separat bewertet werden.

Codebeispiel 5: Die Methode GetRatedDecisionContexts in der Decision Klasse

```
public DecisionContext[] GetRatedDecisionContexts(AIController aiController, float discardThreshold)
{
    // Create contexes according to number of targets
   DecisionContext[] contexts = decisionContextCreator.GetDecisionContexts(this, aiController);

    // Score each context
   for (int i = 0; i < contexts.Length; i++)
   {
      contexts[i].RateContext(considerations, weight, discardThreshold);
   }

   return contexts;
}
```

3.6.6 Consideration

DecisionContexts werden anhand von **Considerations** bewertet. Dies sind ScriptableObjects. Durch die Nutzung von ScriptableObjects kann dieselbe Consideration für mehrere Decisions verwendet werden. Die **Kurven** der Considerations können vom Entwickler intuitiv im Inspektor einer Consideration festgelegt werden.

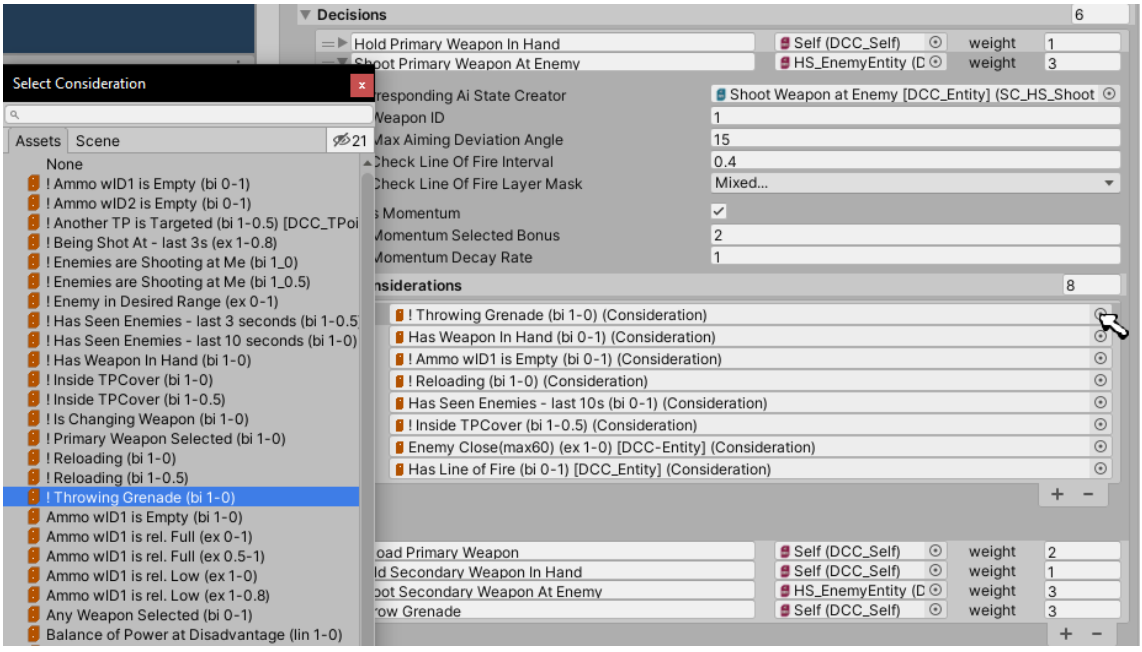


Abbildung 30: Auswahl der Considerations im Inspector der Decision

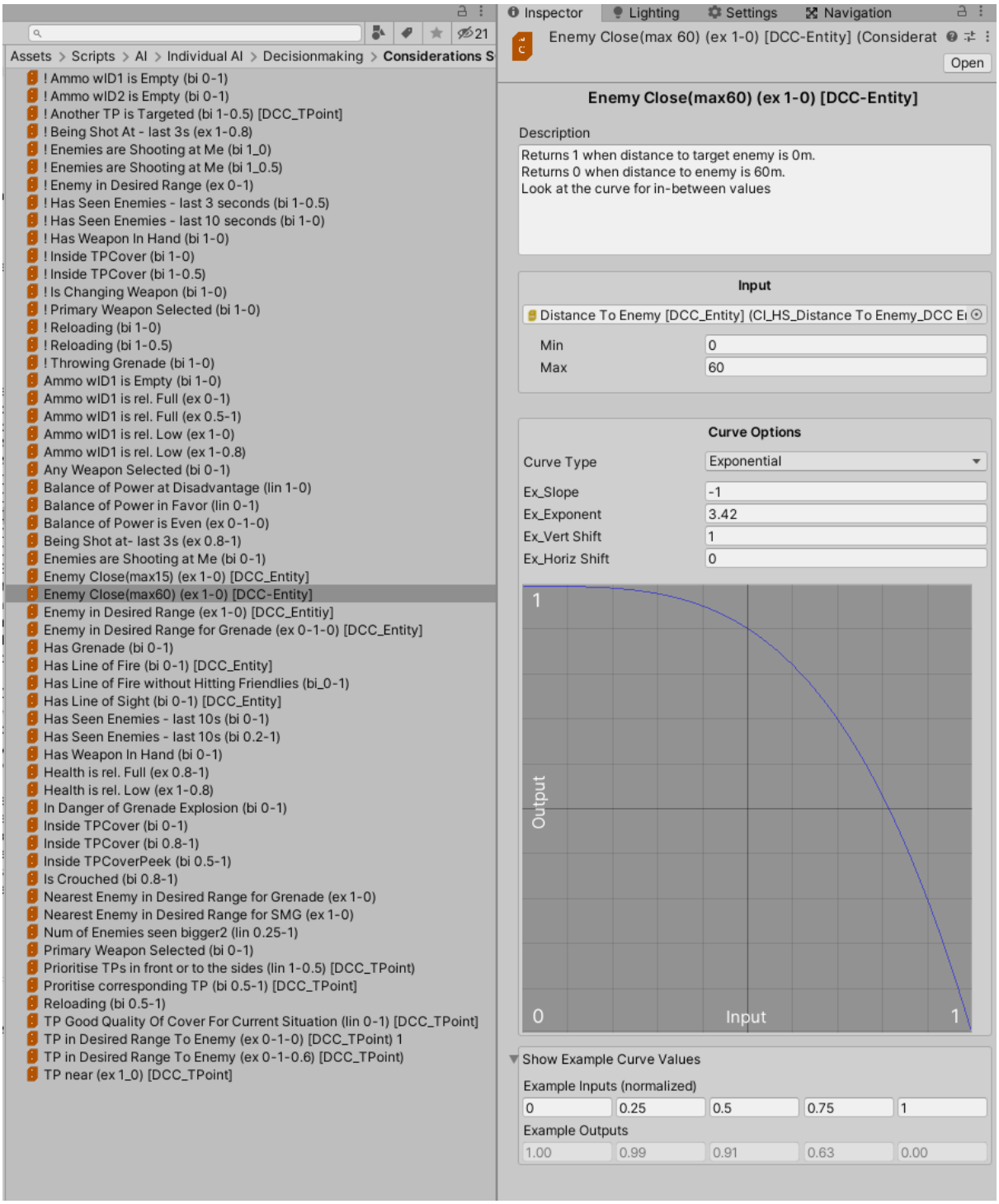


Abbildung 31: Festlegung der Kurve im Inspektor einer Consideration

3.6.7 Kleine Verbesserungen

Discard Treshold

Um die Performance der `Decide()` Methode zu verbessern, wurde ein `DiscardTreshold` eingeführt. Wenn die Bewertung eines `DecisionContexts` unter diesen *Threshold* fällt, wird der `DecisionContext` mit -1 bewertet und die restlichen `Considerations` werden ignoriert. Um möglichst viel aus dieser Mechanik herauszuholen, werden die `Considerations`, die viel zur Bewertung beitragen, ganz vorne in der Liste platziert. `Considerations`, die keinen großen Einfluss auf die Bewertung haben, werden ans Ende der Liste gepackt. Eine Ausnahme bilden `Considerations`, die viel Rechenleistung in Anspruch nehmen, wie z. B. ein Line of Sight - Check. Diese werden auch an das Ende der Liste gepackt, damit sie nur dann ausgeführt werden, wenn dies auch wirklich nötig ist.

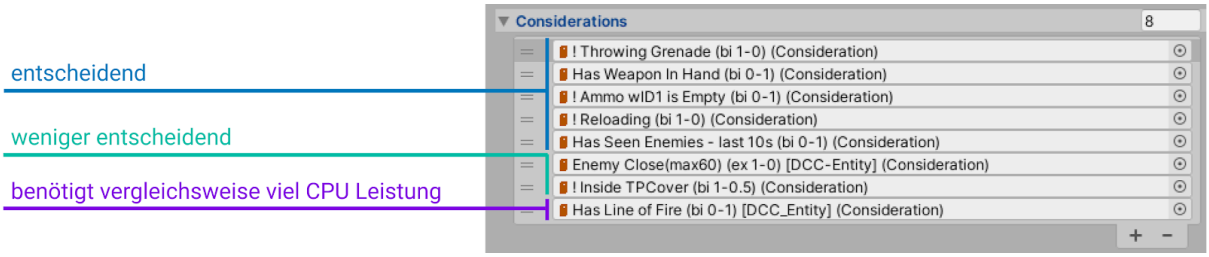


Abbildung 32: Sortieren der Considerations um die Performance zu verbessern

Momentum

Durch eine sich ständig ändernde Spielwelt kann es vorkommen, dass innerhalb einer kurzen Zeit eine große Anzahl an verschiedenen *Decisions* getroffen wird. Eine KI, die sich zu oft umentscheidet und beispielsweise jede Sekunde Ihre Richtung wechselt, kann für den Spieler unter Umständen nicht nachvollziehbar sein.

Um dem entgegenzuwirken, wurde der `Decision` ein *Momentum* hinzugefügt. Dies sind 2 Werte: `momentumSelectedBonus` und `momentumDecayRate`. Sobald ein `DecisionContext` selektiert und ausgeführt wird, wird der Context-Bewertung der Wert des `momentumSelectedBonus` hinzugefügt. Die `momentumDecayRate` stellt die Geschwindigkeit in Bewertung pro Sekunde dar,in welcher dieser Bonus wieder abnimmt.

Dieses Momentum kann für jede `Decision` separat angepasst werden und sorgt dafür, dass sich die KI nach einer getroffenen Entscheidung für die nächsten paar Sekunden sehr wahrscheinlich nicht umentscheiden wird.

Diese scheinbar kleine Veränderung in der Funktionsweise des Systems führt zu einem großen Unterschied in der Wahrnehmung der KI durch den Spieler.

3.7 Optimierung des Sensing und der Individual AI

Das *Sensing* und die `Decide()` Methode der *Individual AI* benötigen viel Rechenleistung. Um die Performance zu verbessern, werden diese Methoden nicht jeden Frame, sondern in einem bestimmten zeitlichen Intervall ausgeführt.

Diese Lösung hat allerdings ein Problem. Wenn beispielsweise alle Soldaten gleichzeitig erstellt werden und dann alle 0.5s ihre Methode ausführen, führt das zu *Spikes* in der Performance. In der folgenden Abbildung ist der Unity-Profiler zu sehen. Dieser zeigt die für das Ausführen des Spiels benötigte Rechenzeit auf der vertikalen Achse und deren Entwicklung über vergangene Frames auf der horizontalen Achse.

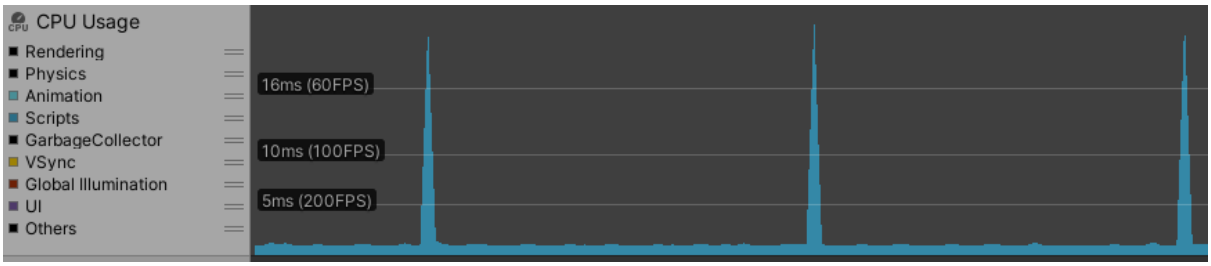


Abbildung 33: Performance bei einem festgelegten Update-Intervall

Unter Umständen würde das Spiel in einem solchen Fall alle 0.5s einen sichtbaren Einbruch der Framerate aufweisen.

Eine schnell implementierte Lösung dieses Problem wäre es, die Zeit des ersten Updates von jedem Soldaten um einen zufälligen Wert zwischen 0 und 0.5s nach hinten zu versetzen. Das Problem mit dieser Lösung ist, dass die zufälligen Werte bei jedem Spielstart anders wären. Einige Spiele hätten die Aufrufe der Methoden gleichmäßig über die Zeit verteilt, andere würden sich hingegen nicht von dem oberen Bild unterscheiden.

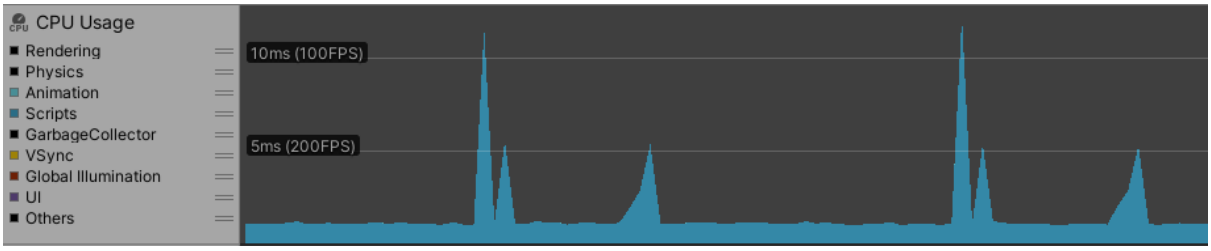


Abbildung 34: Performance bei einem leicht zufälligen Update-Intervall

Eine weitaus zeitaufwendigere, jedoch zuverlässigere Lösung wurde in der Klasse **`ScriptOptimisationManager`** implementiert. Neben einer gleichmäßigen Verteilung der CPU-intensiven Scripts über die Zeit sortiert diese Klasse auch Scripts nach ihrer Wichtigkeit in LOD-Groups.

Für die Optimierung von Scripts müssen verschiedene vom `ScriptOptimisationManager` erbende Klassen erstellt werden. Diese sind Singletons und haben eine Referenz zu allen in der Szene vorhandenen Scripten, welche sie optimieren sollen. Die verschiedenen `OptimisationManager` sortieren alle x Sekunden zu optimierenden Scripte in verschiedene

LOD-Groups. Die Sortierung basiert auf Distanz zu der Kamera des Spielers und der Richtung dieser Kamera. LOD-Groups fassen Scripts von gleicher Priorität zusammen. Beispielsweise werden Objekte direkt vor der Kamera des Spielers alle 20 Frames geupdated, da sie in der LOD0-Group zusammengefasst wurden. Objekte der LOD3-Group, die sich hinter der Kamera befinden und nicht sichtbar sind, werden nur alle 90 Frames geupdated.

Jede dieser LOD-Groups updated die in ihr zusammengefassten Scripts in einem festgelegten Intervall. Dieses Intervall ist in Frames statt in Sekunden angegeben. Die Scripts einer LOD-Group werden gleichmäßig über alle Frames verteilt. Beispielsweise würde eine LOD-Group, welche den Frame-Interval 30 als Parameter hat und 20 Scripts in sich zusammenfasst, jede der ersten 20 Frames genau 1 Script updaten, während es die letzten 10 Frames des Intervalls keine Scripts updaten würde. Anschließend würde sich das wiederholen.

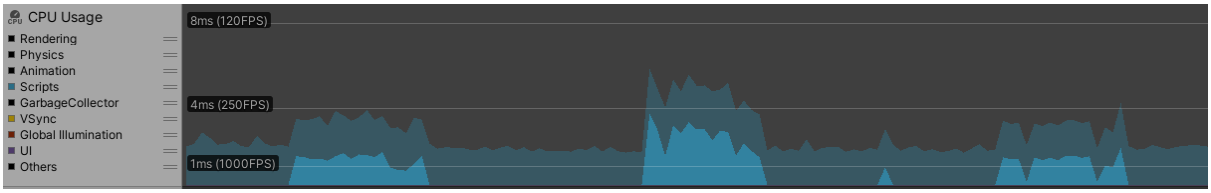


Abbildung 35: Performance mit dem ScriptOptimisationManager

Verteilung der Update Calls (Beispiel)

LOD Group 0:	Update Interval: 10 Frames	Scripts inside Group: 12

Frame 1: update script1, update script11
Frame 2: script2, script12
Frame 3: script3
Frame 4: script4
Frame 5: script5
Frame 6: script6
Frame 7: script7
Frame 8: script8
Frame 9: script9
Frame 10: script10
Frame 11: script1, script 11
Frame 12: script2, script 12
Frame 13: script3
Frame 14: script4
...

Abbildung 36: Verteilung der Update Calls durch den ScriptOptimisationManager

Für die Optimierung der *Individual AI* und des *Sensing* werden die vom `ScriptOptimisationManager` erbenden Klassen **`AIControllerOptimisationManager`** und **`SensingOptimisationManager`** verwendet.

3.8 Evaluierung der Tactical Points

Die Soldaten können sich entscheiden an einem **Cover Point** Deckung zu nehmen oder sich an einem **Cover Peek Point** zu platzieren, um von dort aus zu schießen. Sie müssen sich jedoch entscheiden, welcher *Tactical Point* in ihrer Nähe sich am besten dafür eignet. Neben Aspekten, wie der Entfernung zu diesem *Point,* ist für die KI wichtig, ob dieser sich überhaupt für die aktuelle Situation eignet. Jeder *Tactical Point* hat numerische Bewertungen in 8 Richtungen, welche die Entfernung zur Deckung und die Qualität der Deckung an diesem Punkt beschreiben. Diese Bewertung eines Punktes wird von der KI passend zu der Situation abgelesen und evaluiert. Der *Tactical Point* wird im *Blackboard* des jeweiligen Soldaten auf einer Skala von 0 bis 1 bewertet. Der vorgestellte Algorythmus befindet sich in der Methode `RateTPTogetherWithCorrespondingPoints()` in der Klasse **`AIController_Blackboard`**.

3.8.1 Bewertungsalgorithmus

Die *Tactical Points* werden in Gruppen bewertet. Eine Gruppe bildet immer ein *Cover Point* mit einem oder mehreren dazugehörigen *Cover Peek Points*. Die Bewertung der *Tactical Points* kann nur stattfinden, sobald ein Gegner sichtbar ist und dadurch eine Richtung zu der Gefahr verfügbar ist. Falls dies nicht der Fall ist, werden alle *Tactical Points* mit 0 bewertet.

Bewertung von Cover Points

Für die Bewertung der *Cover Points* spielt die `meanThreatDirection` die entscheidende Rolle. Dies ist ein Vector3 und stellt die Richtung zu allen sichtbaren Gegnern dar. Dieser Vector wird im **`AIController_Blackboard`** berechnet. Bei der Bewertung des *Points* wird geprüft, ob das `DistanceRating` des *Cover Points* in die Richtung der `meanThreatDirection` einen Wert von <2 besitzt. Falls dies der Fall ist, bewertet die KI den CoverPoint mit dem `QualityRating` in die Richtung der meanThreatDirection. Ist das `DistanceRating` >2 bedeutet das, dass die Deckung in dieser Richtung weiter als 2 Meter entfernt ist, was wahrscheinlich nicht optimal ist.

In der Abbildung 37 beträgt die Entfernung zur Deckung in Richtung der `meanThreatDirection` 0,4 Meter und das dazugehörige `QualityRating` beträgt 1,0. Somit würde dieser *Cover Point* für die folgende Situation mit einem Wert von 1,0 bewertet werden.

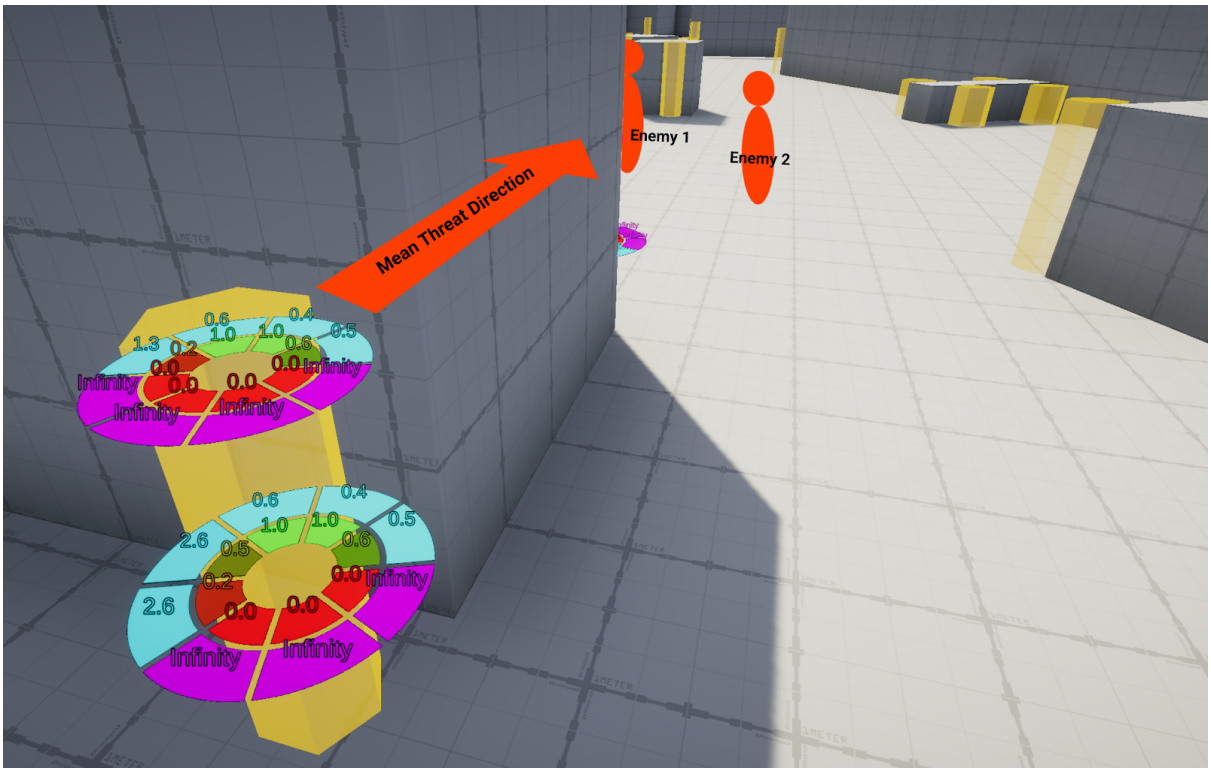


Abbildung 37: Bewertung eines Cover Points

Bewertung von Cover Peek Points

Für die Bewertung eines *Cover Peek Points* spielt neben der `meanThreatDirection` die Distanz zum nächsten gesichteten Gegner eine Rolle. Es wird anhand des `DistanceRatings` des *Cover Peek Points* geprüft, ob die Distanz zur Deckung in die Richtung der `meanThreatDirection` größer ist, als die Distanz zum nächsten Gegner. Falls die Distanz zur Deckung in diese Richtung kleiner wäre, müsste dies bedeuten, dass zwischen dem bewertendem Soldaten und dem nächsten Gegner ein Hindernis steht. Dieses Hindernis könnte das Schießen unmöglich machen, womit sich dieser *Point* nicht zum Schießen auf die Gegner eignen würde.

In der Abbildung 38 ist die Entfernung zu einer Deckung in Richtung der `meanThreatDirection` unendlich und das dazugehörige `QualityRating` spielt keine Rolle. Unendlich ist größer als die Entfernung zum nächsten Gegner, welche 10 Meter beträgt. Dadurch würde dieser *Cover Peek Point* eine Bewertung von 1 bekommen.

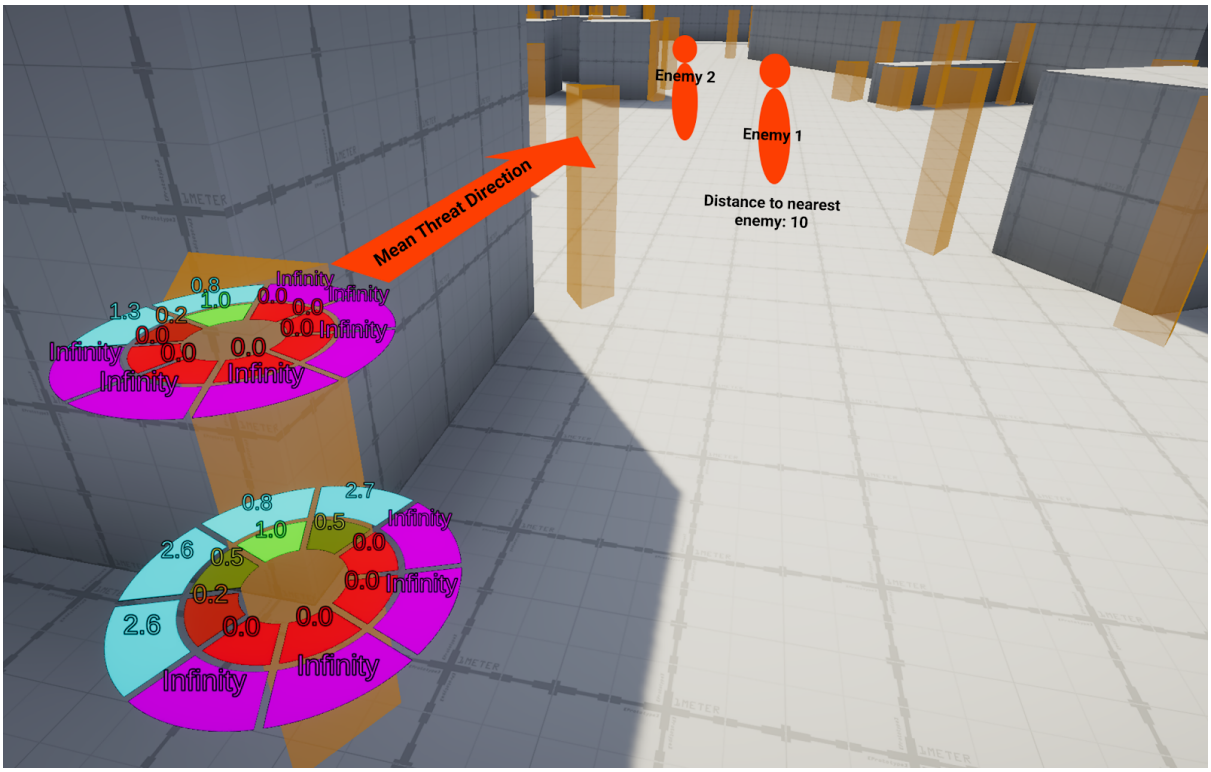


Abbildung 38: Bewertung eines Cover Peek Points

3.8.2 Optimierung der Bewertung

Der Bewertungsalgorithmus braucht verglichen mit anderen Algorithmen in dem Projekt nicht viel Rechenleistung, jedoch wird dieser sehr oft aufgerufen.

Die KI kann bei jedem Aufruf von `Decide()` eine große Anzahl an *Tactical Points* bewerten. In der "Map1" werden beispielsweise von einem Soldaten pro Aufruf der `Decide()` Methode 40 *Tactical Points* bewertet. Wenn sich im Spiel über 20 Soldaten befinden, kann dies zu vielen Bewertungen in kurzer Zeit und dadurch zu Performanceproblemen führen. Um die Performance zu verbessern, werden die *Tactical Points* nur in einem zeitlichen Intervall von 1 Sekunde bewertet. Die Bewertung eines *Points* und dadurch auch die für den Spieler sichtbare Funktionsweise der KI wird sich in einer Sekunde wahrscheinlich nicht drastisch ändern, dadurch lässt sich aber die Anzahl der Aufrufe des Bewertungsalgorithmus verringern.

Die Bewertungen werden in der `Dictionary<int, (float rating, float timeWhenRated)> tPRatingsCache` in der Klasse **`AIController_Blackboard`** gespeichert.

3.9 Squad AI

Aufgrund der mangelnden Zeit zum Ende des Projektes, wurde die *Squad AI* nur auf sehr einfache Weise implementiert. Die Klasse **`ScenarioManager`** besitzt zwei Befehle, die zu Beginn des Spiels den Soldaten aus den 2 verschiedenen Teams erteilt werden.

Die Erteilung der Befehle funktioniert folgendermaßen: Der `ScenarioManager` hat 2 *Decisions*, die jeweils im Inspektor des `ScenarioManagers` durch den Entwickler festgelegt werden. Nachdem ein Soldat gespawnt wird, fügt ihm der ScenarioManager diese *Decision* zu seinen bisher vorhandenen *Decisions* hinzu. Dies passiert mit den Methoden `AddDecision` oder `RemoveDecision` in der Klasse **`AIController`**. Die KI kann sich nun eigenständig entscheiden den erhaltenen Befehl oder andere *Decisions* auszuführen.

Dieses einfache Prinzip des Hinzufügens von *Decisions,* könnte in Zukunft für verschiedene komplexere Squad-Verhalten eingesetzt werden.

Codebeispiel 6: Die Methoden AddDecision und RemoveDecision im AIController

```
public void AddDecision(int decisionLayer, Decision newDecision)
{
   decisionLayers[decisionLayer].AddDecision(newDecision);
}

public void RemoveDecision(int decisionLayer, Decision decisionToRemove)
{
   decisionLayers[decisionLayer].RemoveDecision(decisionToRemove);
}
```

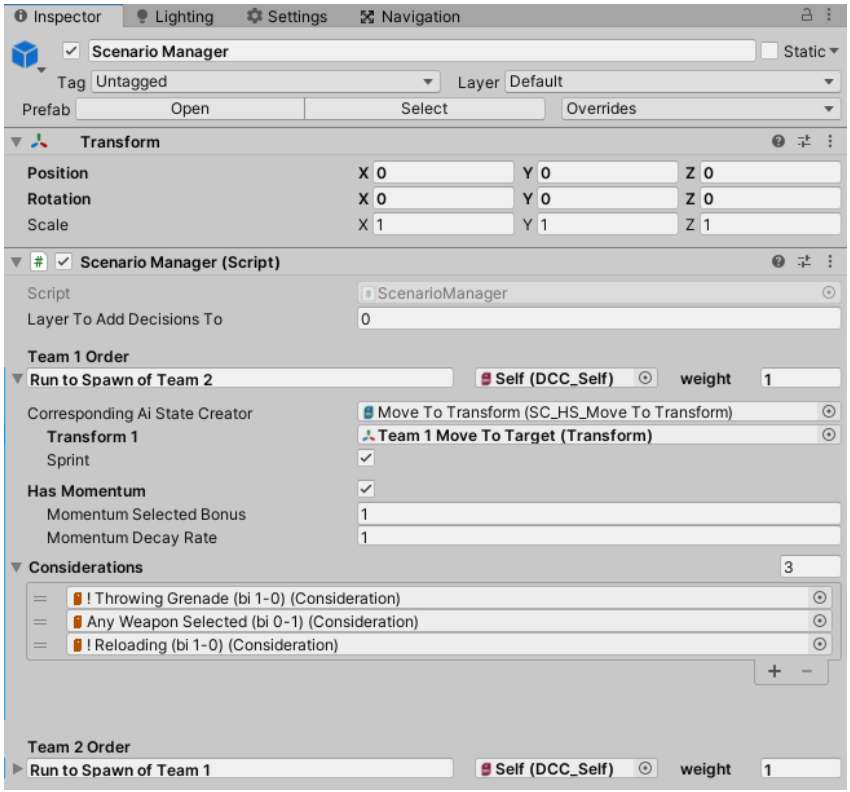


Abbildung 39: Inspektor des ScenarioManagers

4. Auswertung und Erweiterung

4.1 Character Controller

Auswertung

Alle im Konzept vorgestellten Anforderungen an den *Character Controller* wurden implementiert, jedoch ist der *Character Controller* in seinem jetzigen Zustand, meiner Meinung nach nicht zufriedenstellend genug, um in einem fertigen Spiel genutzt zu werden.

Ein Makel des Character Controllers scheinen die Animationen zu sein, welche nicht ganz flüssig und angepasst sind. Animationen eines menschlichen 3D-Charakters waren während des Projekts eine unerwartet große Herausforderung.

Außerdem sind, durch die hohe Komplexität des *Character Controllers*, einige Bugs aufgetreten. Diese sind auf die zu vielen State-Machines verteilt auf alle Sub-Komponenten des *Character Controllers* zurückzuführen. Beispielsweise hören einige Soldaten ab einem bestimmten Punkt auf, Animationen abzuspielen.

Eine der Stärken des *Character Controllers* wäre die Fähigkeit Hindernisse durch Sprünge zu überwinden. Diese Fähigkeit erlaubt dynamische Bewegungen durch die Spielwelt und eröffnet neue Szenarien der Interaktion mit der KI.

Außerdem ist der *Character Controller* als relativ performant einzustufen. Die Implementierung der Animationen und Aktionen eines Charakters erlauben es im jetzigen Stand gleichzeitig ungefähr 100 Soldaten zu simulieren, bevor die Framerate unter 60 fps fällt (Getestet mit einem Intel Core i5-9600KF Prozessor und einer nVidia RTX 2070 Grafikkarte).

Erweiterungsmöglichkeiten

Um eine bessere Qualität der Animationen zu erhalten, würden sich kostspielige fertige Animationen aus digitalen Marktplätzen oder speziell für dieses Projekt angefertigt Animationen besser anbieten.

Um die Prozesse in dem *Character Controller* besser nachzuvollziehen und Bugs zu vermeiden, würde ich in Zukunft die Architektur des *Character Controllers* ändern. Ich würde sie zentralisierter gestalten und mehr Funktionen und States in den `EC_HumanoidCharacterController` statt in dessen Sub-Komponenten verlagern.

4.2 Tactical Points

Auswertung

Das einfache Platzieren der *Points* ist zufriedenstellend. Es nimmt zwar einiges an Zeit in Anspruch, bietet einem Entwickler jedoch viel Kontrolle. Die automatische Berechnung der Qualität per Knopfdruck erspart viel Zeit. Eine manuelle Eingabe dieser Bewertung würde viel Zeit in Anspruch nehmen und würde wenige Vorteile mit sich bringen.

Der Bewertungsalgorithmus selbst liefert beim Finden von passenden Deckungen gute Ergebnisse, allerdings besteht noch Verbesserungsbedarf. Vor allem die Berechnung des *Distance Ratings* liefert oft nicht zufriedenstellende Ergebnisse. Für die ***Open Field Points*** wurde letztendlich keine Verwendung gefunden.

Erweiterungsmöglichkeiten

Das Platzieren der *Points*, welches manuell vorgenommen wird, könnte automatisiert werden. Dies stellt jedoch einerseits eine große Herausforderung dar und andererseits würde ein solch automatisierter Platzierungs-Algorithmus wahrscheinlich nicht in allen Maps/Spielewelten funktionieren. Beispielsweise könnte ein Algorithmus die *Tactical Points* sehr gut in Maps platzieren, die fast ausschließlich aus Gebäuden und Innenräumen bestehen, jedoch hätte dieser Probleme dies zuverlässig auf Maps zu tun, welche ausschließlich aus Wald und Steinen bestehen. Damit ist es sehr wahrscheinlich, dass eine solche automatisierte Platzierung die Vielfalt an Kampfszenarien einschränken würde.

4.3 Sensing

Auswertung

Das Sensing funktioniert überzeugend. Durch das Nutzen der Physik-Layer, ist die benötigte Rechenleistung, verglichen mit dem *Character Controller* oder der *Individual AI,* gering.

Erweiterungsmöglichkeiten

Das *Sensing*-System könnte in Zukunft realistischer gestaltet werden. Beispielsweise könnte man das Hören verbessern. Aktuell werden alle Gegner in einem bestimmten Radius automatisch “gehört”, obwohl im Spiel keine Geräusche existieren.

Außerdem könnten Soldaten erkennen, dass ein gesichteter verbündeter Soldat in einem Kampf verwickelt ist. Daraus könnten sie deduzieren, dass sich vor dem verbündeten Soldaten ein Gegner befindet, obwohl dieser noch nicht sichtbar ist. Eine einfache Variante dieses Mechanismus wurde zum Ende des Projektes hin implementiert und kann mit dem UI Button “Copy Enemy Infos from Friendlies” aktiviert werden.

Abbildung 40: Copy Enemy Infos from Friendlies Button

4.4 Individual AI

Auswertung

Das von Dave Mark in Vorträgen vorgestellte “Infinity Axis Utility System” wurde vollständig und funktionsfähig implementiert und um einige eigene Elemente erweitert. Die Performance dieses Systems hat meine Erwartungen übertroffen.

Ein großer Vorteil dieses Systems ist die Flexibilität der Entscheidungen. In bestimmten Fällen kann dies aber auch als Nachteil gesehen werden. Im Vergleich zu anderen, in Spielen üblichen Architekturen wie “Behaviour Trees” oder “Finite State Machines” hat ein Designer bei dem Utility System weniger Kontroller über die Entscheidungen der KI. Dadurch können mehr unerwartete Situationen entstehen, welche abhängig vom Szenario gewollt oder ungewollt sein können. Außerdem ist es nicht immer klar, wie ein bestimmtes Verhalten zu erzielen ist, da für jede Entscheidung sehr viele kleine Faktoren eine Rolle spielen. Viele KI-Designer müssten in diesem Fall umdenken und der KI mehr “freie Hand” gewähren, ohne jeden ihrer Schritte kontrollieren zu wollen.

Erweiterungsmöglichkeiten

Verbesserung der Performance

Im aktuellen Projekt werden viele *Considerations* unnötigerweise mehrmals evaluiert. Wenn ein Soldat sich entscheiden will in einem *Tactical Point* Deckung zu nehmen, kann es unter Umständen dazu kommen, dass er 20 *Tactical Points* zur Auswahl hat und dadurch 20 verschiedene *Decision Contexts* bewerten muss. In diesem 20 *Decision Contexts* bezieht sich aber nur die Hälfte der *Considerations* auf die *Tactical Points*. Die Hälfte der *Considerations* hat 20-mal denselben Wert, da sie sich auf den Soldaten selber beziehen.

Um diesen unnötigen Berechnungen entgegenzuwirken, könnte statt der Liste an *Decisions* im *Decision Maker* eine baumartige Struktur verwendet werden. Dadurch würden bestimmte *Decisions* nur infrage kommen, sobald vorher festgelegte *Considerations* eine bestimmte Bewertung erreicht haben.

Ein weiterer Decision Maker

Aktuell können Entscheidungen der KI durch die UI nachvollzogen werden. In einem fertigen first-person Shooter wäre dies aber nicht auf diese Weise möglich. Eine sehr gute Art die Entscheidungen einer KI für den Spieler nachvollziehbar zu machen sind Sounds.

Ein weiterer *Decision Maker* könnte erstellt werden, der sich um zur Situation passende Sounds kümmert. Als Reaktion auf Schaden könnte der Soldat aufschreien, er könnte seinen Verbündeten sagen, dass er nachlädt und daher Deckung braucht oder auch seine Gegner beleidigen und dadurch menschlicher erscheinen. Jede *Decision* in diesem *Sound-Decision Maker* würde basierend auf Ihren *Considerations* einen dieser Sounds abspielen.

4.5 Erweiterungsmöglichkeiten für die Squad AI

Der aktuelle Stand der *Squad AI* ist sehr rudimentär. Die Soldaten bekommen von dem `ScenarioManager` lediglich den Befehl zu der anderen Seite der Map zu laufen. Der `ScenarioManager` kann in diesem Fall als hierarchisch übergeordnete Squad-KI gesehen werden.

Basierend auf diesem einfachen Prinzip des Hinzufügens von *Decisions*, könnten in Zukunft auch weitaus komplexere Squad-Verhalten implementiert werden. Im Folgenden einige Beispiele:

Beispiel 1: Fliehen

Szenario

Der Spieler zusammen mit einigen verbündeten Soldaten kämpft in einer Halle gegen ein Team von gegnerischen Soldaten. Sobald das gegnerische Team 70% ihrer Soldaten verliert, fliehen sie.

Mögliche Implementierung

Ein speziell dafür angefertigtes *Scenario-Management-Script* müsste implementiert werden. Dieses würde eine Referenz zu allen gegnerischen Soldaten besitzen. Alle x Sekunden würde dieses *Szenario-Management-Script* überprüfen, wie viele Soldaten noch am Leben sind. Falls dies weniger als 30 % sind, wird jedem der noch lebenden feindlichen Soldaten die *Decision* “Flee” zu ihren `AIControllern` hinzugefügt. Die *Decision* “Flee” ist möglichst hoch gewichtet, sodass sich alle Soldaten entscheiden, diese auszuführen.

Beispiel 2: Taktisches Vorrücken

Szenario

Ein Squad von 4 Soldaten befindet sich in Deckung. Der Squad möchte in die Richtung der vermuteten Gefahr vorrücken. Während sich 3 Soldaten zu einer Deckung näher in die Richtung der Gefahr bewegen, bleibt der 4 Soldat zurück und gibt den vorrückenden Soldaten Deckung. Anschließend warten die 3 vorderen Soldaten, dass der 4 Soldat ihnen folgt. Sobald der 4 Soldat die Gruppe einholt, wiederholt sich das Prozedere.

Mögliche Implementierung

Ein *Squad-Manager Script* müsste hierfür implementiert werden. Dieses würde eine Referenz zu den 4 Soldaten besitzen. Der *Squad-Manager* hätte 2 *Decisions* festgelegt. Eine *Decision* wäre “Rücke vor zur Deckung in Richtung der Gefahr” und die andere wäre “Bleibe in Deckung”.

Basierend auf festgelegten Konditionen würde der *Squad-Manager* einem der 4 Soldaten die *Decision* “Bleibe in Deckung” hinzufügen, während die 3 anderen die *Decision* “Rücke vor zur Deckung in Richtung der Gefahr” erhalten. Da diese hinzugefügten Decisions hoch gewichtet wären, würden sie wahrscheinlich sofort ausgeführt werden.

Zu einem beliebigen Zeitpunkt könnte der Squad Manager die hinzugefügten Decisions wieder aus den `AIControllern` der Soldaten entfernen, um das ursprüngliche Verhalten der Soldaten wiederherzustellen.

Beispiel 3: Verteidigung eines taktisch wichtigen Ortes

Szenario

Ein bestimmter Ort in der Spielwelt ist für das Kampfgeschehen wichtig. Eine Gruppe von Soldaten soll sich zu diesem Ort bewegen und ihn bis zum Tode verteidigen. Dieser Ort könnte beispielsweise ein gut positionierter Bunker sein.

Mögliche Implementierung

Ein speziell dafür angefertigtes *Scenario-Management-Script* müsste implementiert werden. Dies hätte Referenzen zu den in Frage kommenden Soldaten und zu den *Tactical Points,* welche sich in dem in Frage kommenden Bunker befinden. Das *Szenario-Management-Script* würde den Soldaten die *Decision* hinzufügen "Gehe zum Tactical Point". Diese *Decision* hätte einen der *Tactical Points* im Bunker als Ziel und wäre im Vergleich zu den anderen Decisions dieser Soldaten sehr hoch gewichtet.

4.6 Debug- Visalisierung

Auswertung

Die Debug-UI bietet einen guten Einblick in das Verhalten der KI und vereinfacht das Nachvollziehen dieser Arbeit. Hier können vorher getroffenen *Decisions* vom Entwickler aufs Detail analysiert werden. Die UI hat das Weiterentwickeln der KI auf jeden Fall beschleunigt, da Fehler sehr schnell erkennbar sind.

Da diese UI in einem fertigen Spiel keine Verwendung finden würde, wurde die Visualisierung schnell und nicht besonders sorgfältig implementiert. Das sorgt dafür, dass die UI in manchen Situationen mehr Rechenleistung in Anspruch nimmt, als alle anderen im Projekt vorhandenen Systeme und dadurch die Performance unter 60 fps fallen kann.

Erweiterungsmöglichkeiten

In der Zukunft würde ich die erstellte UI komplett entfernen und jegliche Visualisierungen im EditMode oder im Inspector durchführen. Dies wäre einfacher und performanter.

5. Fazit

In der vorliegenden Arbeit wurde eine KI für einen 3D first-person Shooter entwickelt.

Die dabei verwendeten animierten menschlichen Charaktere bieten eine überzeugende Approximation zu Gegnern in einem komplett ausgebauten Shooter, jedoch war deren Entwicklung unerwartet schwer.

Eine Abstrahierung der Umgebungsinformationen sorgt dafür, dass die Agenten ihre Deckung in den meisten Fällen aus taktischer Sicht passend wählen können. Die Methode, Punkte in der Welt in 8 Richtungen zu bewerten, scheint aufgrund der dadurch entstehenden Flexibilität gut gewählt zu sein.

Für die KI der einzelnen Charaktere wurde, das “Infinity Axis Utility System” genutzt. Ein Pluspunkt dieses Systems ist der hohe Grad an Flexibilität. Das Verhalten der KI kann im Vergleich zu anderen üblichen KI-Architekturen, mit wenig Aufwand verändert werden. Die KI kann unter Umständen sogar ganz ohne Programmierkenntnisse designt werden, sobald die passenden *Considerations*, *AIStates* und ein *Character Controller* bereits programmiert wurde. Auch kann die Grundstruktur dieser KI ohne Veränderung für andere Spiele verwendet werden. Beispielsweise wäre diese KI meiner Meinung nach auch gut für Tiere in einem Zoo Simulator oder Monster und Dorfbewohner in einem RPG geeignet.

Eine den Charakteren hierarchisch übergeordnete Squad-KI wurde aufgrund mangelnder Zeit nur auf einfachste Weise implementiert, doch sie scheint vielversprechend und würde den ersten Schritt in der Weiterentwicklung dieser Arbeit darstellen.

Mit besseren Animationen und Modellen und mehr Zeit für das Designen und Testen der passenden *Decisions* und *Considerations*, kann ich mir sehr wohl vorstellen, dass diese KI ohne große Veränderungen für ein kommerzielles Spiel genutzt werden könnte. Die Performance und Möglichkeiten wären dafür meiner Meinung nach ausreichend.

6. Verwendete Materialien

Tools

Unity Engine https://unity.com/

Blender https://www.blender.org/

Photoshop https://www.adobe.com/products/photoshop.html

FMOD https://www.fmod.com/ (letztendlich nicht genutzt)

Assets

Navmesh Components https://github.com/Unity-Technologies/NavMeshComponents

Look Animator https://assetstore.unity.com/packages/tools/animation/look-animator-122249

Resonance Audio https://resonance-audio.github.io/resonance-audio/ (letztendlich nicht genutzt)

SPR Batcher Profiler https://github.com/Unity-Technologies/SRPBatcherBenchmark/blob/master/Assets/Scripts/SRPBatcherProfiler.cs

Mixamo - Animationen und ein Charaktermodell https://www.mixamo.com/#/

Graphy https://assetstore.unity.com/packages/tools/gui/graphy-ultimate-fps-counter-stats-monitor-debugger-105778

Navmesh Cleaner https://assetstore.unity.com/packages/tools/ai/navmesh-cleaner-151501

Kopierter Code

Two Bone IK Resolver https://wirewhiz.com/how-to-code-two-bone-ik-in-unity/

Hiding or Disabling inspector properties using PropertyDrawers http://www.brechtos.com/hiding-or-disabling-inspector-properties-using-propertydrawers-within-unity-5/

ShowWhen Attribute https://gist.github.com/deebrol/02f61b7611fd4eca923776077b92dfc2

Quaternion Helper Methods https://gist.github.com/maxattack/4c7b4de00f5c1b95a33b

7. Quellenverzeichnis

---------- **Vorträge, die das “Inifinite Axis Utility System” vorstellen** ----------

AI Summit. (2010). *Improving AI Decision Modeling Through Utility Theory*. GDC Vault. **https://www.gdcvault.com/play/1012410/Improving-AI-Decision-Modeling-Through**

AI Summit. (2012). *Embracing the Dark Art of Mathematical Modeling in AI*. GDC Vault. **https://gdcvault.com/play/1015683/Embracing-the-Dark-Art-of**

AI Summit. (2013). *Architecture Tricks: Managing Behaviours in Time, Space and Depth*. GDC Vault. **https://www.gdcvault.com/play/1018040/Architecture-Tricks-Managing-Behaviors-in**

AI Summit. (2015). *Building a Better Centaur at Massive Scale*. GDC Vault. **https://www.gdcvault.com/play/1021848/Building-a-Better-Centaur-AI**

----------**Analyse erfolgreicher Spiele**----------

Horti, Samuel. (2017, April 03). *Why F.E.A.R.'s AI is still the best in first-person shooters.* Rock Paper Shotgun. **https://www.rockpapershotgun.com/why-fears-ai-is-still-the-best-in-first-person-shooters**

[lupianwolf] (2017, April 03). *Why F.E.A.R.’s AI is still the best in first-person shooters*. [Online forum post]. Retrieved from **https://www.reddit.com/r/Games/comments/639a43/why_fears_ai_is_still_the_best_in_first person/**

Orkin, Jeff. (2016). *Three States and a Plan: The A.I. of F.E.A.R.* [PDF] **https://alumni.media.mit.edu/~jorkin/gdc2006_orkin_jeff_fear.pdf**

[LESkidd113] (2016, Juli 16). *The A.I. of F.E.A.R. Paper.* [Online forum post]. Retrieved from **https://www.reddit.com/r/gamedev/comments/8zaybb/the_ai_of_fear_papergdc_2006/**

Straatman Remco| Van der Sterren, William | Beij Arjen. (2005, Mai 10). *Killzone’s AI: dynamic procedural combat tactics.* [PDF]

https://www.cgf-ai.com/docs/straatman_remco_killzone_ai.pdf

Straatman Remco| Van der Sterren, William. (2005, Mai 10). *Killzone's AI: dynamic procedural combat tactics.* [PDF] **https://www.cgf-ai.com/docs/killzone_ai_gdc2005_slides.pdf**

[Tipotas688] (2019, Februar 04). *FSM, BT, HTN, Goap, other.* [Online forum post]. Retrieved from **https://www.gamedev.net/forums/topic/700989-fsm-bt-htn-goap-other/**

AI and Games. (2015, Mai 11). *The Behaviour Tree AI of Halo 2 | AI and Games* [Video]. Youtube. **https://www.youtube.com/watch?v=NU717sd8oUc&ab_channel=AIandGames**

AI and Games. (2020, Mai 06). *Building the AI of F.E.A.R. with Goal Oriented Action Planning | AI 101.* [Video]. Youtube. **https://www.youtube.com/watch?v=PaOLBOuyswI&ab_channel=AIandGames**

AI and Games. (2019, Mai 25). *The AI of Half-Life: Finite State Machines | AI 101.* [Video]. Youtube. **https://www.youtube.com/watch?v=JyF0oyarz4U&ab_channel=AIandGames**

AI and Games. (2018, Januar 31). *The AI of Shogun: Total War (Part 1 of 5) | AI and Games* .[Video]. Youtube. **https://www.youtube.com/watch?v=XBzTJOYgW0M&ab_channel=AIandGames**

AI and Games. (2018, Februar 05). *The AI of Empire: Total War (Part 2 of 5) | AI and Games* .[Video]. Youtube. **https://www.youtube.com/watch?v=XBzTJOYgW0M&ab_channel=AIandGames**

AI and Games. (2018, Februar 05). *The AI of Empire: Total War (Part 2 of 5) | AI and Games* [Video]. Youtube. **https://www.youtube.com/watch?v=KL_AAGSivbI&ab_channel=AIandGames**

AI and Games. (2018, Februar 12). *The Campaign AI of Total War: Rome II (Part 3 of 5) | AI and Games* [Video]. Youtube. **https://www.youtube.com/watch?v=1m9-7ZrpbBo&ab_channel=AIandGames**

AI and Games. (2018, Februar 19). *The Diplomacy AI in Total War: Attila (Part 4 of 5) | AI and Games* [Video]. Youtube. **https://www.youtube.com/watch?v=UsA368WHUcE&ab_channel=AIandGames**

AI and Games. (2018, Februar 26). *The Siege Battle AI of Total War: Warhammer (Part 5 of 5) | AI and Games* [Video]. Youtube. **https://www.youtube.com/watch?v=zPsvSDJkejA&ab_channel=AIandGames**

GDC. (2019, Juli 11). *Authored vs. Systemic: Finding a Balance for Combat AI in Uncharted 4* [Video]. Youtube. **https://www.youtube.com/watch?v=G8W7EQKBgcg&ab_channel=GDC**

GDC. (2016, Oktober 08). *Less is More: Designing Awesome AI for Games* [Video]. Youtube. **https://www.youtube.com/watch?v=1xWg54mdQos&ab_channel=GDC**

----------Andere----------

Mark, Dave. (2012, November 01). *AI Architectures: A Culinary Guide (GDMag Article).* Intrinsic Algorithm. **http://intrinsicalgorithm.com/IAonAI/2012/11/ai-architectures-a-culinary-guide-gdmag-article/**

AI and Games. (2018, Oktober 07). *How A Navigation Mesh Works in 3D Games | AI 101* [Video]. Youtube. **https://www.youtube.com/watch?v=U5MTIh_KyBc&list=PLokhY9fbx05eeUZCNUbelL-b0TyVizPjt&index=6&ab_channel=AIandGames**

AI and Games. (2019, Januar 02). *Behaviour Trees: The Cornerstone of Modern Game AI | AI 101* [Video]. Youtube. **https://www.youtube.com/watch?v=6VBCXvfNlCM&ab_channel=AIandGames**

AI Summit. (2010). *Little Big AI: Rich Behavior on a Small Budget*. GDC Vault. **https://www.gdcvault.com/play/1012413/Little-Big-AI-Rich-Behavior**

AI Summit. (2012). *Believable Tacticas for Squad AI*. GDC Vault. **https://www.gdcvault.com/play/1015665/Believable-Tactics-for-Squad**

Rasmussen, Jacob. (2016, April 27). *Are Behavior Trees a Thing of the Past?.* Gamasutra. **https://www.gamasutra.com/blogs/JakobRasmussen/20160427/271188/Are_Behavior_Trees_a_Thing_of_the_Past.php**

AI Summit. (2010). *Deciding on an AI Architecture: Which Tool for the Job?* GDC Vault.

https://www.gdcvault.com/play/1012411/Deciding-on-an-AI-Architecture

AI Summit. (2012). *Situational Awareness: Terrain Reasoning for Tactical Shooter AI.* GDC Vault. **https://www.gdcvault.com/play/1015718/Situational-Awareness-Terrain-Reasoning-for**

AI Summit. (2012). *Less A More I: Using Psychology in Game AI.* GDC Vault. **https://www.gdcvault.com/play/1015682/Less-A-More-I-Using**

AI Summit. (2013). *The Simplest AI Trick in the Book.* GDC Vault. **https://www.gdcvault.com/play/1018059/The-Simplest-AI-Trick-in**

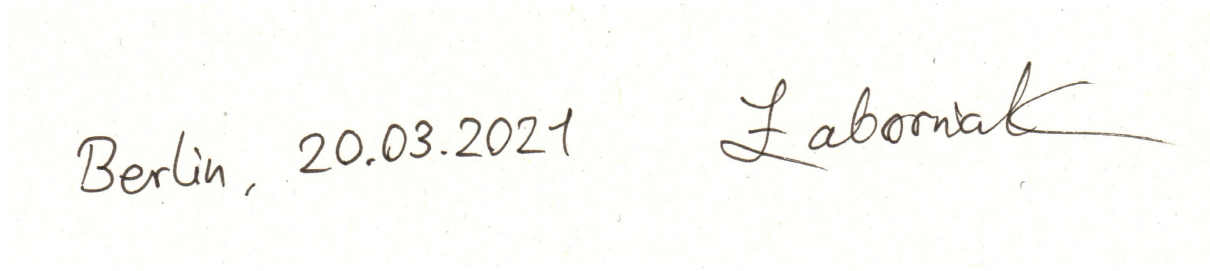
8. Abbildungsverzeichnis

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe.

Alle sinngemäß und wörtlich übernommenen Textstellen aus fremden Quellen wurden kenntlich gemacht.

Diese Arbeit wurde in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegt.



Anhang: Liste aller dem Soldier zur Verfügung stehender Decisions

Decision Maker 1: Positioning Layer

Run Away From Grenade Threat

DecisionContextCreator: Self
AIStateCreator: Run Away From Grenade
Weight: 8

Considerations:

In Danger of Grenade Explosion (bi 0-1)

Go to TPCover - Walk

DecisionContextCreator: HS_TPointCover
AIStateCreator: Move to TPoint
Weight: 6

Considerations:

Has Seen Enemies - last 10s (bi 0-1)
! Throwing Grenade (bi 1-0)
TP in Desired Range To Enemy (ex 0-1-0) [DCC_TPoint)
TP near (ex 1_0) [DCC_TPoint]
Ammo wID1 is rel. Low (ex 1-0.8)
Being Shot at- last 3s (ex 0.8-1)
Health is rel. Low (ex 1-0.8)
! Another TP is Targeted (bi 1-0.5) [DCC_TPoint]
Prioritise TPs in front or to the sides (lin 1-0.5) [DCC_TPoint]
Proritise corresponding TP (bi 0.5-1) [DCC_TPoint]
TP Good Quality Of Cover For Current Situation (lin 0-1) [DCC_TPoint]

Go to TPCoverPeek - Walk

DecisionContextCreator: HS_TPointCoverPeek
AIStateCreator: Move to TPoint
Weight: 6

Considerations:

! Is Changing Weapon (bi 1-0)
Has Seen Enemies - last 10s (bi 0-1)
TP in Desired Range To Enemy (ex 0-1-0) [DCC_TPoint)
TP near (ex 1_0) [DCC_TPoint]
Ammo wID1 is rel. Full (ex 0-1)
Health is rel. Full (ex 0.8-1)
! Being Shot At - last 3s (ex 1-0.8)
! Another TP is Targeted (bi 1-0.5) [DCC_TPoint]
Prioritise TPs in front or to the sides (lin 1-0.5) [DCC_TPoint]
Proritise corresponding TP (bi 0.5-1) [DCC_TPoint]
TP Good Quality Of Cover For Current Situation (lin 0-1) [DCC_TPoint]

Hold Position Combat Stance Crouching

DecisionContextCreator: Self
AIStateCreator: Hold Position
Weight: 2

Considerations:

Has Seen Enemies - last 10s (bi 0-1)
Reloading (bi 0.5-1)
Being Shot at- last 3s (ex 0.8-1)
Nearest Enemy in Desired Range for SMG (ex 1-0)

Hold Position Combat Stance Standing

DecisionContextCreator: Self
AIStateCreator: Hold Position
Weight: 2

Considerations:

Has Seen Enemies - last 10s (bi 0-1)
! Being Shot At - last 3s (ex 1-0.8)
! Reloading (bi 1-0.5)
Nearest Enemy in Desired Range for SMG (ex 1-0)

Decision Maker 1: Item Interaction Layer

Hold Primary Weapon In Hand

DecisionContextCreator: Self
AIStateCreator: Hold Weapon Scan for Threat
Weight: 1

Considerations:

! Throwing Grenade (bi 1-0)
! Ammo wID1 is Empty (bi 0-1)
! Has Seen Enemies - last 3 seconds (bi 1-0.5)

Shoot Primary Weapon At Enemy

DecisionContextCreator: HS_EnemyEntity
AIStateCreator: Shoot Weapon at Enemy
Weight: 3

Considerations:

! Throwing Grenade (bi 1-0)
Has Weapon In Hand (bi 0-1)
! Ammo wID1 is Empty (bi 0-1)
! Reloading (bi 1-0)
Has Seen Enemies - last 10s (bi 0-1)
Enemy Close(max60) (ex 1-0) [DCC-Entity]
! Inside TPCover (bi 1-0.5)
Has Line of Fire (bi 0-1) [DCC_Entity]

Reload Primary Weapon

DecisionContextCreator: Self
AIStateCreator: Reload Weapon
Weight: 2

Considerations:

! Throwing Grenade (bi 1-0)
Has Weapon In Hand (bi 0-1)
Ammo wID1 is rel. Low (ex 1-0)
Inside TPCover (bi 0.8-1)
Is Crouched (bi 0.8-1)
! Being Shot At - last 3s (ex 1-0.8)

Hold Secondary Weapon In Hand

DecisionContextCreator: Self
AIStateCreator: Hold Weapon Scan for Threat
Weight: 1

Considerations:

! Throwing Grenade (bi 1-0)
Ammo wID1 is Empty (bi 1-0)
! Has Seen Enemies - last 3 seconds (bi 1-0.5)

Shoot Secondary Weapon At Enemy

DecisionContextCreator: HS_EnemyEntity
AIStateCreator: Shoot Weapon at Enemy
Weight: 3

Considerations:

! Throwing Grenade (bi 1-0)
Ammo wID1 is Empty (bi 1-0)
! Ammo wID2 is Empty (bi 0-1)
Has Seen Enemies - last 10s (bi 0-1)
! Reloading (bi 1-0)
! Inside TPCover (bi 1-0.5)
Enemy Close(max15) (ex 1-0) [DCC_Entity]
Has Line of Fire (bi 0-1) [DCC_Entity]

Throw Grenade

DecisionContextCreator: Self
AIStateCreator: Throw Grenade
Weight: 3

Considerations:

Has Grenade (bi 0-1)
! Reloading (bi 1-0)
Has Seen Enemies - last 10s (bi 0.2-1)
Num of Enemies seen bigger 2 (lin 0.25-1)
Inside TPCoverPeek (bi 0.5-1)
! Enemies are Shooting at Me (bi 1_0.5)
Nearest Enemy in Desired Range for Grenade (ex 1-0)