



HOCHSCHULE HEILBRONN

# ENTWICKLUNG EINES PHYSIKBASierten CHARAKTERCONTROLLERS MIT UNITY ML AGENTS

**Software-Engineering**  
Fakultät für Informatik  
der Hochschule Heilbronn

## **Bachelor-Thesis**

vorgelegt von

**Simon Grözing**  
Matrikelnummer: 205047

# Inhaltsverzeichnis

<b>1. Einleitung</b>	<b>6</b>
<b>2. Grundlagen</b>	<b>7</b>
2.1. Verstärkendes Lernen . . . . .	7
2.2. ML-Agents . . . . .	8
2.2.1. Aufbau . . . . .	8
2.2.2. Komponenten . . . . .	9
2.2.3. Programmierschnittstellen . . . . .	10
2.3. Unity Physik . . . . .	12
<b>3. Analyse</b>	<b>13</b>
3.1. Szenenaufbau . . . . .	13
3.2. Physikkomponenten und -konfiguration . . . . .	14
3.3. Agent implementierung . . . . .	15
3.4. Ziel . . . . .	16
<b>4. Fazit</b>	<b>17</b>
<b>A. Anhang 1</b>	<b>18</b>

# Abbildungsverzeichnis

2.1.	Verstärkendes Lernen Ablauf . . . . .	7
2.2.	Unity ML-Agents Aufbau . . . . .	8
2.3.	Unity ML-Agents Verhalten Komponente . . . . .	9
2.4.	Unity ML-Agents Entscheidung Anfragen Komponente . . . . .	10
3.1.	Walker-Demo Hierarchy . . . . .	13
3.2.	Agent Hierarchy . . . . .	14
3.3.	Agent Konfiguration . . . . .	16

# Tabellenverzeichnis

2.1. Test . . . . .	10
---------------------	----

# Listings

2.1. Academy Instanzvariablen . . . . .	10
2.2. Agent Funktionen . . . . .	11
2.3. Trainer Konfigurationsdatei . . . . .	11

# 1. Einleitung

Machine Learning Modelle bieten neue Möglichkeiten den Prozess der Charakter animation zu erleichtern. In der Thesis soll ein Ansatz anhand bestehender Literatur und Beispiele erforscht werden, in dem Spielcharaktere physikalisch mit Rigidbodies und Joints simuliert und mit Hilfe von Machine Learning trainiert werden, um möglichst realistische Bewegung nachahmen zu können.

## 2. Grundlagen

Dieses Kapitel behandelt die Grundlagen der verwendeten Technologien, Paketen und Unity Komponenten.

### 2.1. Verstärkendes Lernen

Der Begriff 'Verstärkendes Lernen' beschreibt eine Art von Problemstellung und die dafür geeigneten Problemlösungsmethoden im Bereich des Maschinellen Lernens. Die grundlegenden Bestandteile einer Trainingsumgebung sind der Agent und die Umgebung, in der der Agent seine Aktionen ausführt. Der Ansatz ist in vielerlei Hinsicht vergleichbar mit dem Lernvorgang von Menschen. Ein Baby lernt das Krabbeln ohne direkte Anweisungen, nur durch die Wahrnehmung der Umgebung, das Verhalten der Umgebung in Relation zu seinen Bewegungen und die mit den Bewegungen einhergehenden Belohnungen. Auf dieselbe Art lernt der Agent beim Verstärkenden Lernen von jedem Zustand die Aktion auszuführen, um die Belohnung zu maximieren. Die Belohnung können dabei positiv oder negativ sein. Im Fall des Babys sind die Belohnungen Faktoren wie Schmerz, Hunger, Müdigkeit oder gestillte Neugier. Der Agent hingegen erhält eine numerische Belohnung.[2]

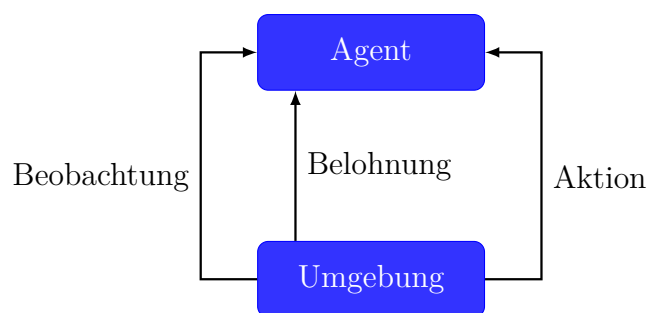


Abbildung 2.1.: Verstärkendes Lernen Ablauf

Die Abbildung 2.1 zeigt die Verbindungen zwischen dem Agenten und der Umgebung. Der Agent erhält als Input einen Zustand oder meist einen Teilzustand der Umgebung und reagiert darauf mit einer Aktion. Dieser Zyklus kann je nach Problem in unterschiedlichen

Intervallen durchlaufen werden. Bei kontinuierlichen Kontrollproblemen werden Aktionen meist in regelmäßigen Intervallen abgefragt. Bei rundenbasierten Spielen kann dieser Vorgang jedoch auch nur einmal pro Runde stattfinden.

## 2.2. MI-Agents

Das Unity ML-Agents Toolkit ist ein Open-Source-Projekt, in dem maschinelle Lernalgorithmen und Funktionen für die Verwendung mit der Spieleumgebung Unity implementiert und kontinuierlich weiterentwickelt werden.

### 2.2.1. Aufbau

Die Implementierung ist in zwei Bereiche unterteilt. Für die Unity-Integration ist das Paket `com.unity.ml-agents` aus dem Unity Asset Store zuständig. Das eigentliche Training mit den maschinellen Lernalgorithmen findet jedoch in einer separaten Python-Umgebung statt. Für die Kommunikation zwischen den beiden Bereichen verwendet das ML-Agents Toolkit eine gRPC-Netzwerkkommunikation, worüber Zustand der Simulationsumgebung in Unity, ausgewählte Aktionen des neuronalen Netzes in Python und weitere Werte für die Auswertung des Trainings ausgetauscht werden.[1]

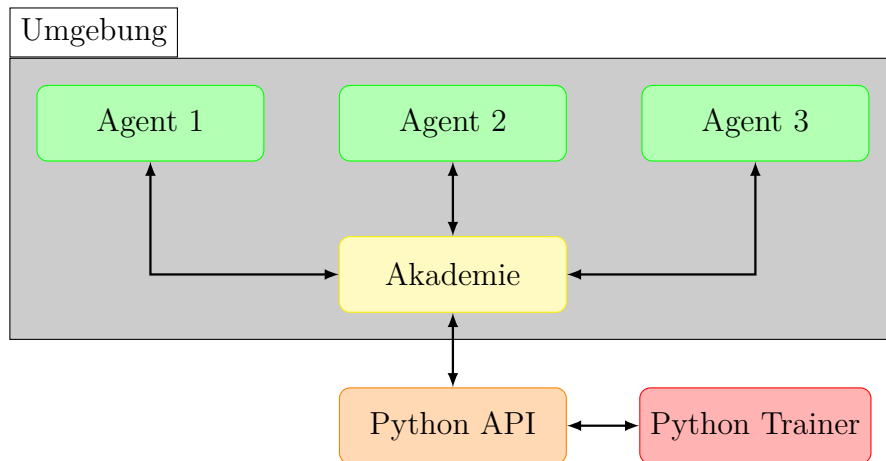


Abbildung 2.2.: Unity ML-Agents Aufbau

Das Unity-Paket enthält drei Grundlegenden Komponenten, die Akademie, Agenten und Sensoren. Das Unity-Paket enthält zwei Komponenten: Agenten und deren Verhalten. Die Agent-Komponente bildet die Grundlage für alle Implementierungen. Sie bietet abstrakte Funktionen für die Initialisierung, den Start einer Episode, das Erfassen des Zustands der



Umgebung sowie das Ausführen von Aktionen. Durch die Implementierung dieser Funktionen können unterschiedlichste Agenten entwickelt und trainiert werden. Jeder Agent ist mit einem Verhalten verknüpft, das für jede Beobachtung des Agenten eine Aktion auswählt, die der Agent ausführt. Es gibt drei Arten, wie die Verhaltensweisen agieren können. Im Lernmodus werden die Beobachtungen des Agenten für das Training und die Auswahl einer Aktion anhand des aktuellen Modells verwendet. Der Inferenzmodus nutzt hingegen ein bereits trainiertes Modell und wertet dieses aus. Der letzte Modus eines Verhaltens ist der Heuristikmodus, bei dem festgelegte Regeln im Code entscheiden, welche Aktion ausgeführt wird, ohne die Verwendung eines trainierten Modells.[1]

### 2.2.2. Komponenten

In diesem Kapitel werde Ich die Grundlegenden Komponenten des Unity ML-Agents Packets, welche in der Arbeit verwendet wurden erklären. Dadurch sollten Codeausschnitte und Komponentenabbildungen in folgenden Kapiteln deutlich zu verstehen sein.

#### Verhalten

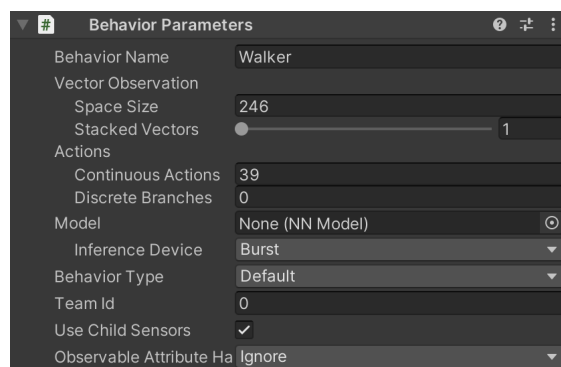


Abbildung 2.3.: Unity ML-Agents Verhalten Komponente

Konfigurationsfeld	Beschreibung
Behaviour Name	Name des Verhaltens / wird in Trainer Konfiguration referenziert
Space Size	Anzahl an Beobachtungen / Inputknoten für NN
Continuous Actions	Anzahl an Aktionen / Outputknoten von NN
Model	Referenz auf bereits trainiertes Modell zur Verwendung in Inferenz
Behaviour Type	Lernmodus Default = Lernen, Heuristic, Inferenz

Tabelle 2.1.: Test

## Entscheidung

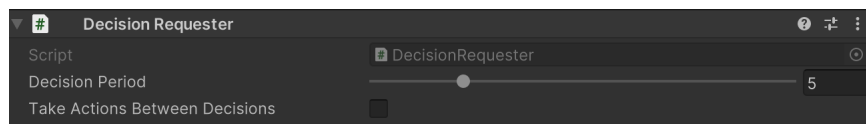


Abbildung 2.4.: Unity ML-Agents Entscheidung Anfragen Komponente

Konfigurationsfeld	Beschreibung
Decision Period	Anzahl an Akademie-Schritten (standard ein Schritt pro Physikupdate) bis zur nächsten Entscheidung
Take Actions Between Decisions	Kontrollkasten ob Agent Aktionen zwischen Entscheidungen ausführen soll

### 2.2.3. Programmierschnittstellen

```

1 envParams = Academy.Instance.EnvironmentParameters;
2 statsRecorder = Academy.Instance.StatsRecorder;

```

Listing 2.1: Academy Instanzvariablen

Die Akademie stellt mit dem Attribut `EnvironmentParameters` die Umgebungsparameter aus Trainer Konfiguration oder aktueller Lektion bereit

Mit dem `StatsRecorder` lassen sich Daten aggregieren um diese nach oder während dem Training über die Tensorboard Visualisierung auszuwerten

```

1 public override void CollectObservations(VectorSensor sensor)
2 {
3     sensor.AddObservation(floatObservation);
4 }
5
6 public override void OnActionReceived(ActionBuffers actionBuffers)
7 {
8     var continuousActions = actionBuffers.ContinuousActions;
9     float action = continuousActions[0]
10 }
11
12 public virtual void FixedUpdate()
13 {
14     AddReward(floatReward);
15 }

```

Listing 2.2: Agent Funktionen

In der `CollectObservations` Methoden wird festgelegt welche Daten dem Agent für das Training bereit stehen, dieser Schritt wird für jede angefragte Entscheidung ausgeführt und das Ergebnis an das NN Modell oder den Python Trainer übergeben.

Wenn eine Entscheidung angefragt wurde und das NN Modell ein Ergebnis liefert wird dieses hier von numerischen Werten in Aktionen umgewandelt.

Im folgenden Beispielcode wird ein Reward in jedem `FixedUpdate` vergeben über die `AddReward` Methode die auch Teil der Agenten-Komponente ist. Der Reward kann aber an jeder Stelle im Code vergeben werden, der Code dient hier nur als ein Beispiel.

Die Trainings Konfigurationsdatei enthält mehrere Teile. Der hyperparameter Teil enthält die Hyperparameter des Maschinellen Lernalgorithmuses, danach folgt der `network_settings` Teil welcher die Konfiguration des Neuronalelernetzes festlegt. Anschließend folgen noch Konfigurationen für die Belohnungssignale im Bereich `reward_signals` und Einstellungen für die Speicherung der Daten sowie der länge des Trainings. Ganz am Ende der Konfigurationsdatei befinden sich noch Umgebungsparameter welche erweitert und während dem Training ausgelesen werden können.

```

1 {
2 behaviors:
3   Walker:
4     trainer_type: ppo
5     hyperparameters:
6       batch_size: 2048
7       buffer_size: 20480
8       learning_rate: 0.0003

```

```
9      beta: 0.005
10     epsilon: 0.2
11     lambda: 0.95
12     num_epoch: 3
13     learning_rate_schedule: linear
14 network_settings:
15     normalize: true
16     hidden_units: 256
17     num_layers: 3
18     vis_encode_type: simple
19 reward_signals:
20     extrinsic:
21         gamma: 0.995
22         strength: 1.0
23 keep_checkpoints: 5
24 checkpoint_interval: 5000000
25 max_steps: 30000000
26 time_horizon: 1000
27 summary_freq: 30000
28 environment_parameters:
29     environment_count: 100.0
30 }
```

Listing 2.3: Trainer Konfigurationsdatei

## 2.3. Unity Physik

## 3. Analyse

Zusätzlich zu den maschinellen Lernkomponenten liefert Unity auch Demonstrationsumgebungen, in denen verschiedene Lösungen für gängige Verstärkungslernprobleme implementiert sind. In der Walker-Demo wird ein physisch simulierter Charakter darauf trainiert, zu einem Zielwürfel zu laufen. Diese Demo-Umgebung implementiert bereits einige Grundlagen für die Steuerung eines physisch simulierten Charakters. Aus diesem Grund wird in dieser Arbeit die Walker-Demo als Grundlage für die Entwicklung genutzt. In diesem Kapitel wird daher die Walker-Demo analysiert, um in den folgenden Kapiteln darauf aufzubauen.

### 3.1. Szenenaufbau

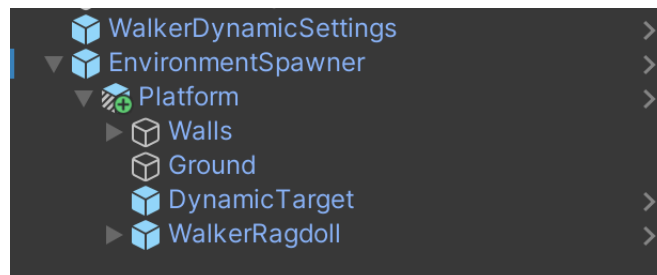


Abbildung 3.1.: Walker-Demo Hierarchy

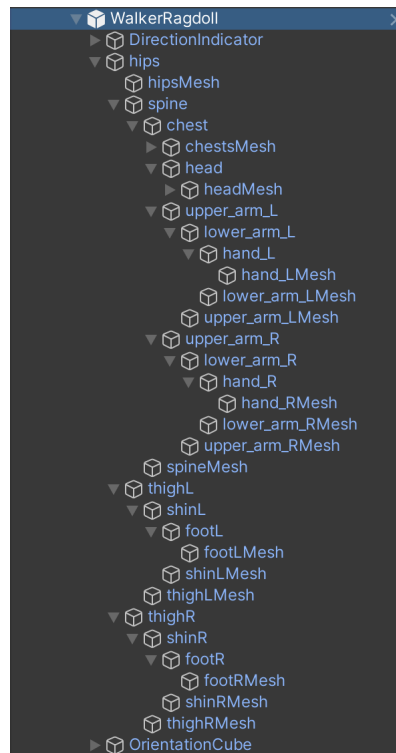


Abbildung 3.2.: Agent Hierarchy

## 3.2. Physikkomponenten und -konfiguration

Der Körper besteht aus 11 Kapseln, drei Kugeln und 2 Quadern, jeder dieser Formen hat eine Festkörper und eine Kollisions Physikkomponente. Zwischen den Körperteilen werden die Gelenke als Kugelgelenke simuliert.

Körperteil	Verbundenes Körperteil	Gewicht	Winkellimits	Form
Hüfte	-	15kg	-	Kapsel
Wirbelsäule	Hüfte	10kg	$x(-20,20) \ y(-20,20) \ z(-15,15)$	Kapsel
Oberkörper	Wirbelsäule	8kg	$x(-20,20) \ y(-20,20) \ z(-15,15)$	Kapsel
Kopf	Oberkörper	6kg	$x(-30,10) \ y(-20,20)$	Kugel
Oberarm LR	Oberkörper	je 4kg	$x(-60,120) \ y(-100,100)$	Kapsel
Unterarm LR	Oberarm	je 3kg	$x(0,160)$	Kapsel
Hand LR	Unterarm	je 2kg	-	Kugel
Oberschenkel LR	Hüfte	je 14kg	$x(-90,60) \ y(-40,40)$	Kapsel
Unterschenkel LR	Oberschenkel	je 7kg	$x(0,120)$	Kapsel
Fuß LR	Unterschenkel	je 5kg	$x(-20,20) \ y(-20,20) \ z(-20,20)$	Quader

### 3.3. Agent implementierung

lernablauf (Beobachtung, Aktionen ausführen, Belohnungsfunktion, einrichtung)

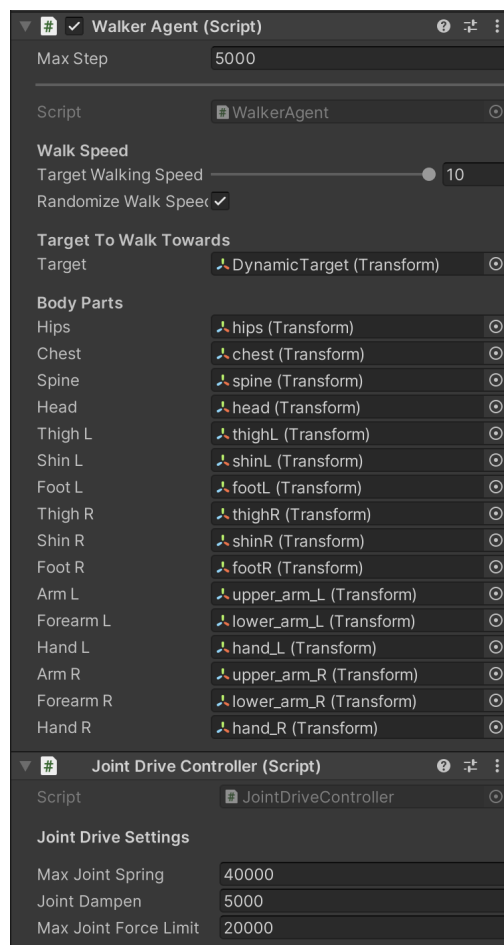


Abbildung 3.3.: Agent Konfiguration

Agent Code hier einfügen? oder evtl. im Anhang?

## 3.4. Ziel

Ziele (target controller)



## 4. Fazit

Text

## **A. Anhang 1**

# Literaturverzeichnis

- [1] Arthur Juliani u. a. “Unity: A general platform for intelligent agents”. In: *arXiv preprint arXiv:1809.02627* (2020). URL: <https://arxiv.org/pdf/1809.02627.pdf>.
- [2] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.