Entwicklung eines physikbasierten Charaktercontrollers mit Unity ML Agents

Software-Engineering
Fakultät für Informatik
der Hochschule Heilbronn

Bachelor-Thesis

vorgelegt von

Simon Grözinger
Matrikelnummer: 205047

Inhaltsverzeichnis

Abbildungsverzeichnis

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. 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. Im Fall des Babys sind die Belohnungen Faktoren wie Schmerz, Hunger, Müdigkeit oder gestillte Neugier. Der Agent hingegen erhält eine numerische Belohnung.[2]

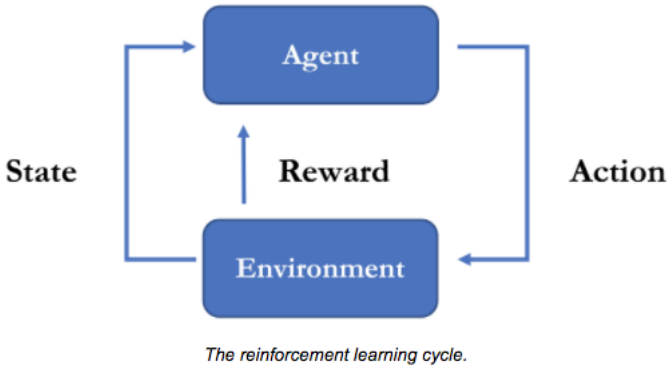

Abbildung 2.1.: Verstärkendes Lernen Ablauf [3]

Die Abbildung 2.1 zeigt die Verbindungen zwischen dem Agent 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.

3. PPO

Das Unity ML-Agents Packet beinhaltet zwei bereits implementierte Algorithmen, SAC (Soft Actor Critic) und PPO (Proximal Policy Optimization). Beim testen der Walker Demo mit den zwei Algorithmen waren die Endergebnisse vergleichbar. Jedoch hat der SAC Algorithmus in dieser Umgebung mehr Zeit gebraucht und das Lernverhalten war instabiler. Daher wird in der Arbeit der PPO Algorithmus von Open Ai verwendet. In diesem Kapitel wird die Funktionsweise hinter dem Algorithmus näher erklärt.

Der PPO Algorithmus basiert auf dem Akteur Kritik (Actor Critic) Ansatz. Der Akteur Kritik Ansatz nutz zwei neuronale Netze, das Akteur Netz lernt die Zuordnung von Zustand zu Aktion während das Kritik Netz bewertet wie gut es ist die Aktion zu wählen basierend auf dem aktuellen Zustand.

Akteur Verlust Funktion:

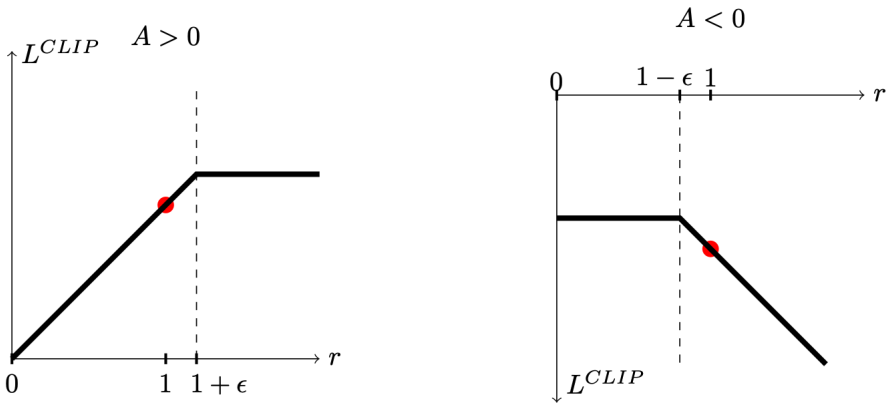

Abbildung 3.1.: PPO Clip Funktion [1]

```
1 ratio = torch.exp(logp - logp_old)
2 clip_adv = torch.clamp(ratio, 1-clip_ratio, 1+clip_ratio) * adv
3 loss_pi = -(torch.min(ratio * adv, clip_adv)).mean()
```

Listing 3.1: Codeausschnitt von Akteur Verlustfunktion aus Open Ai Spinning Up PPO implementation

Kritik Verlust Funktion: Um die Abweichung zwischen der Vorhersage des Kritik Netzes und dem tatsächlichen Ertrag zu messen nutzt PPO die mittlere quadratische Abweichung.

```
1 return ((ac.v(obs) - ret)**2).mean()
```

Listing 3.2: Codeausschnitt von Kritik Verlustfunktiion aus Open Ai Spinning Up PPO implementation

4. 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. 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 C# Kommunikator-Klasse, die über gRPC-Netzwerkkommunikation mit dem Python-Prozess kommuniziert. Der Python-Prozess kommuniziert über die Python Low-Level-API, die die Kommunikation übernimmt und die Befehle an den Trainer weiterleitet.[6]

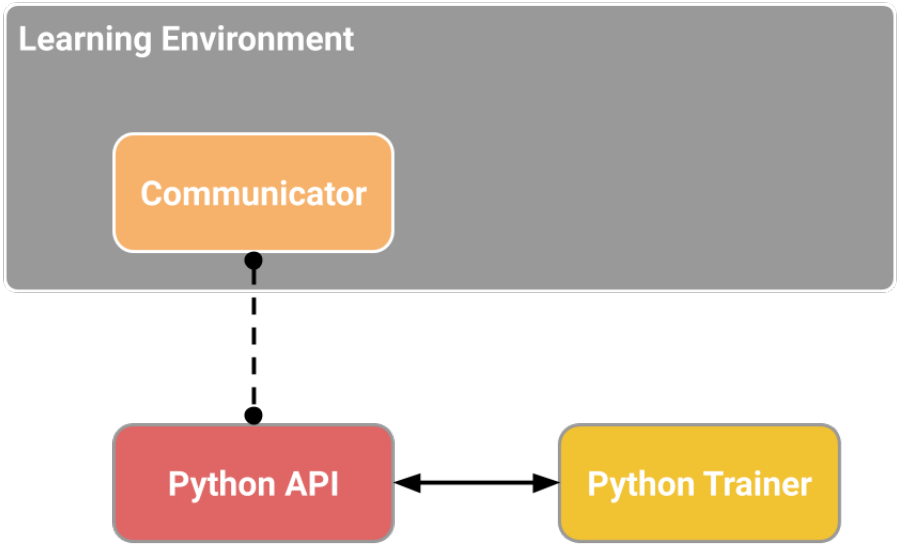

Abbildung 4.1.: Unity ML-Agents Lernumgebung [4]

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 Ver-

halten 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.[6]

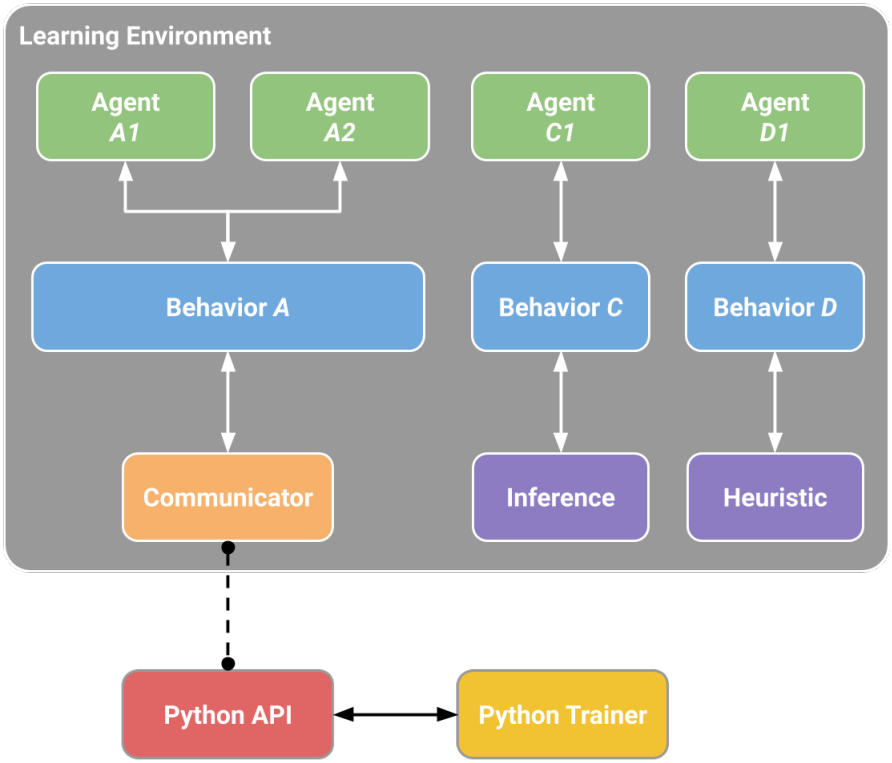

Abbildung 4.2.: Unity ML-Agents Lernumgebung Beispiel [5]

4.1. Komponenten

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

4.1.1. Verhalten

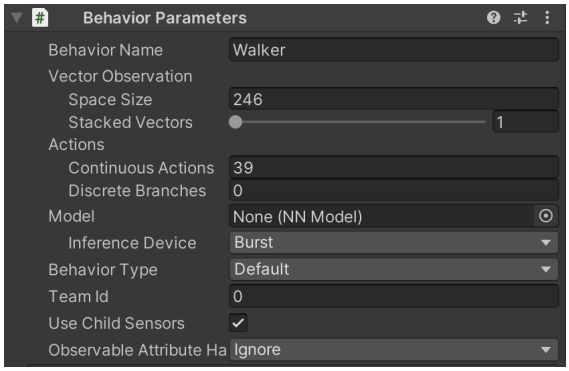

Abbildung 4.3.: Unity ML-Agents Verhalten Komponente

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

4.1.2. Entscheidung

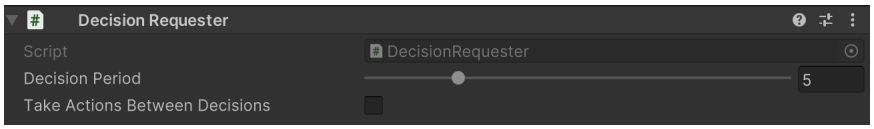

Abbildung 4.4.: Unity ML-Agents Entscheidung Anfragen Komponente

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

4.1.3. Agent Abstrakte Funktionen

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

```
envParams = Academy.Instance.EnvironmentParameters;
```

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

```
statsRecorder = Academy.Instance.StatsRecorder;
```

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.

```
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(floatObservation);
}
```

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

```
public override void OnActionReceived(ActionBuffers actionBuffers)
{
    var continuousActions = actionBuffers.ContinuousActions;
    float action = continuousActions[0]
}
```

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.

```
public virtual void FixedUpdate()
{
    AddReward(floatReward);
}
```

Die Trainings Konfigurationdatei enthält mehrere Teile. Der hyperparameter Teil enthält die Hyperparameter des Maschinellen Lernalgorithmuses, danach folgt der network_settings Teil welcher die Konfiguration des Neuronalennetzes 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.

```
{
behaviors:
  Walker:
    trainer_type: ppo
    hyperparameters:
      batch_size: 2048
      buffer_size: 20480
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: true
      hidden_units: 256
      num_layers: 3
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.995
        strength: 1.0
    keep_checkpoints: 5
    checkpoint_interval: 5000000
    max_steps: 30000000
    time_horizon: 1000
    summary_freq: 30000
environment_parameters:
  environment_count: 100.0
}
```

5. 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.

5.1. Szenenaufbau

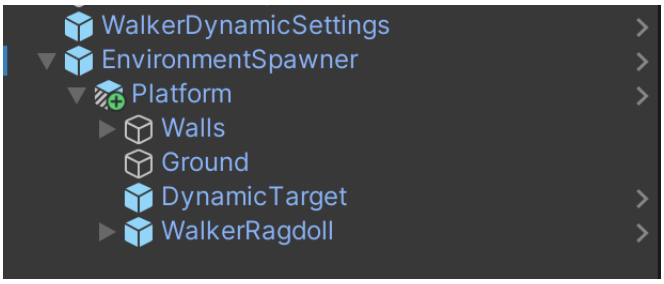

Abbildung 5.1.: Walker-Demo Hierarchy

Abbildung 5.2.: Agent Hierarchy

5.2. 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

5.3. Agent implementierung

lernablauf (Beobachtung, Aktionen ausführen, Belohnungsfunktion, einrichtung)

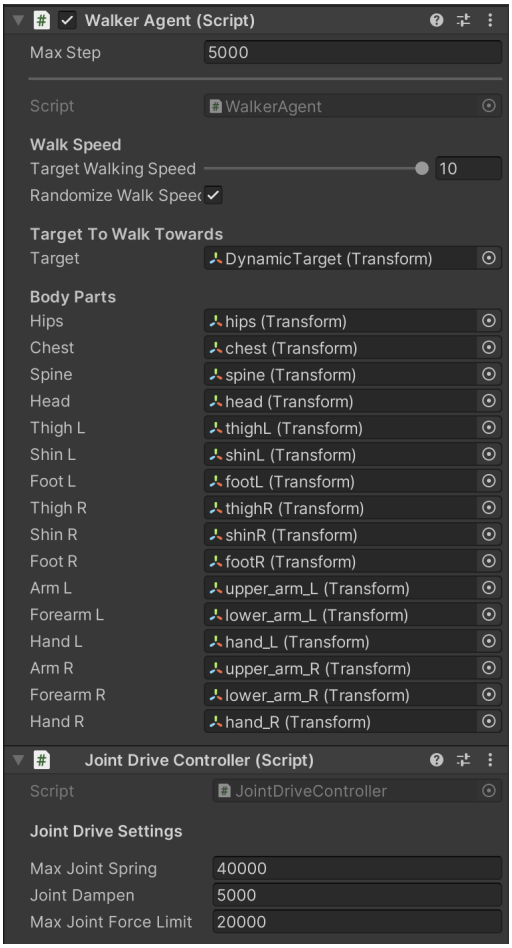

Abbildung 5.3.: Agent Konfiguration

Agent Code hier einfügen? oder evtl. im Anhang?

5.4. Ziel

Ziele (target controller)

6. Fazit

Text

A. Anhang 1

Literaturverzeichnis

[1] John Schulman u. a. “Proximal policy optimization algorithms”. In: *arXiv preprint arXiv:1707.06347* (2017).

[2] Richard S Sutton und Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[3] Unity. *rl cycle*. 27.02.2018. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/images/rl_cycle.png.

[4] Unity. *unity mlagents learning environment*. 29.04.2020. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/images/learning_environment_basic.png.

[5] Unity. *unity mlagents learning environment example*. 29.04.2020. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/images/learning_environment_example.png.

[6] Unity. *unity mlagents toolkit overview*. 6.06.2023. URL: https://github.com/Unity-Technologies/ml-agents/blob/release_21/docs/ML-Agents-Overview.md.