Projektdokumentation

Web- und Interaction Design

AI-Evolution

Unity-Anwendung mit Machine Learning

Daniel Depta

Abgabedatum: 19.02.2023

Fachhochschule Erfurt

Kurzfassung

Ziel dieser Arbeit ist es, menschliche Eingaben mit den erlernten Eingaben einer *künstlichen Intelligenz* (KI) zu vergleichen. Dafür wurde eine Unity-Anwendung entwickelt, die ein *Machine Learning*-Framework einbindet, um ein Objekt (Rakete) zu steuern.

Die Steuerung soll dabei leicht erlernbar sein, jedoch auch für erfahrene Benutzer zu Beginn eine Herausforderung darstellen. Dadurch sollen die Präzision und die Reaktionsgeschwindigkeit der künstlichen Intelligenz ersichtlich werden.

Dazu wurde zuerst zu verschiedenen Themengebieten recherchiert und eine Planung durchgeführt. Darauf aufbauend wurde eine Unity-Anwendung aufgesetzt und die Steuerung implementiert. Als nächstes wurde das Machine Learning-Framework *ML-Agents* eingebunden, konfiguriert und in den Unity-Skripten angesteuert. Danach konnte die KI trainieren und die Anwendung angepasst werden, wenn Probleme auftraten.
Abschließend wurde eine neue Szene erstellt, in der zwei steuerbare Raketen vorhanden sind: eine für den menschlichen Spieler und eine für die KI. So können beide gegeneinander antreten.

Inhaltsverzeichnis

Abbildungsverzeichnis

1 Motivation

Zum einen war die Motivation hinter diesem Projekt, sich mit *Machine Learning* (vor allem in Spielen) auseinanderzusetzen und dies auszuprobieren. Zum anderen wollte ich experimentell herausfinden, ob ich innerhalb eines abgesteckten Rahmes eine künstliche Intelligenz schaffen kann, die sich besser als ein Mensch anstellt.

Die gesammelten Erfahrungen können eventuell in zukünftigen Projekten angewendet werden, beispielsweise um zukünftig künstliche Intelligenz zu implementieren. Außerdem werden KIs immer populärer, daher ist es für mich hilfreich und interessant, mich damit auch experimentell auseinanderzusetzen, statt nur Berichte darüber zu lesen.
Insgesamt kann der Vergleich der Steuerung von Menschen mit der von künstlichen Intelligenzen in Computerspielen dazu beitragen, das Verständnis von Mensch-Maschine-Interaktionen und die Entwicklung von künstlicher Intelligenz zu verbessern.

2 Idee

2.1 Erster Ansatz – Laufroboter

Es musste zuerst ein passendes mögliches Szenario gefunden werden, welches innerhalb eines Semesters umsetzbar ist. Daher sollte die umzusetzende Anwendung schlicht gehalten werden.

Meine erste Idee war es, ein zweidimensionales Spiel zu entwickeln, in welchem ein vereinfachter Roboter mit zwei menschenähnlichen Beinen das Laufen erlernt. Dazu war geplant, dass die Bewegung über Hüft- und Kniegelenke ermöglicht wird.

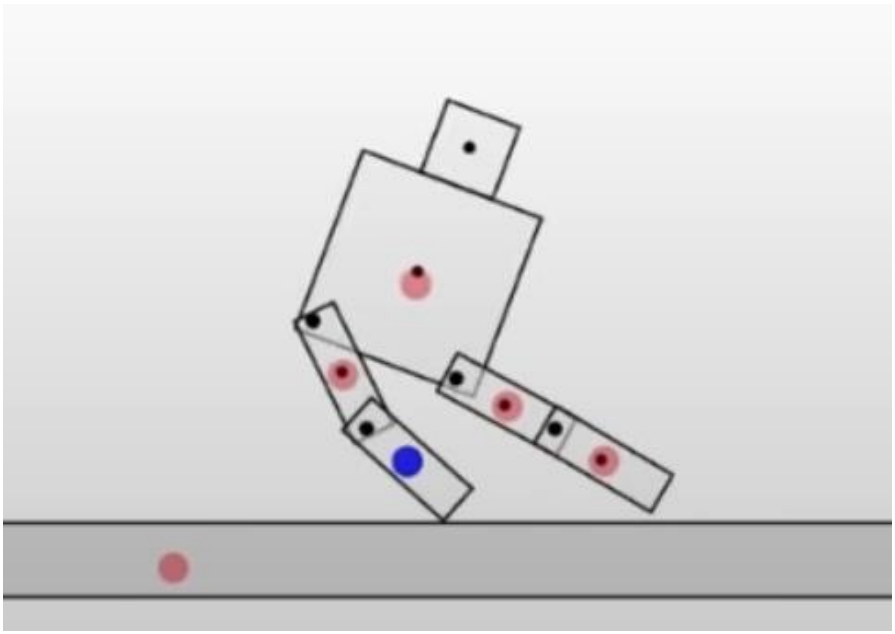


Abbildung 2-1: Screenshot aus Video "A.I. Learns To Walk" – 2D-Roboter

Das hätte den Vorteil, dass der Roboter sein Gleichgewicht nur in zwei Achsen statt in drei Achsen halten muss. Das wäre für die KI vermutlich leichter gewesen. Jedoch ist mir für dieses Szenario keine Lösung eingefallen, wie die Gelenke durch menschliche Eingaben sinnvoll gesteuert werden können. Für präzise Steuerung wären wahrscheinlich Joysticks nützlich, um die Drehgeschwindigkeiten kontrollieren zu können. Dies wäre voraussichtlich sehr schwer erlernbar gewesen.

Außerdem wurde vorgeschlagen, ein Szenario in einer dreidimensionalen Welt zu finden, weil dies für menschliche Bedienung interessanter ist.

2.2 Finale Idee – Rakete

Daher habe ich meine ursprüngliche Idee angepasst, sodass sie in einer 3D-Umgebung anwendbar ist: statt Laufen soll das Schweben/Fliegen erlernt werden.
Die Idee ist, eine Art Rakete mit vier Schubdüsen zu erschaffen. Wenn alle Schubdüsen aktiviert werden, sollte die Rakete gerade nach oben abheben. Wenn weniger Düsen aktiv sind, soll die Rakete in eben diese Richtung geschoben werden.

Da das Spielfeld auch einen Boden enthalten soll, ist ein dauerhafter Flug nicht notwendig. Mit drei aktiven Düsen sollte die Rakete also über die Bodenfläche schleifen können. Mit gezielten Gegenbewegungen soll es auch möglich sein, die Rakete im Gleichgewicht zu halten.

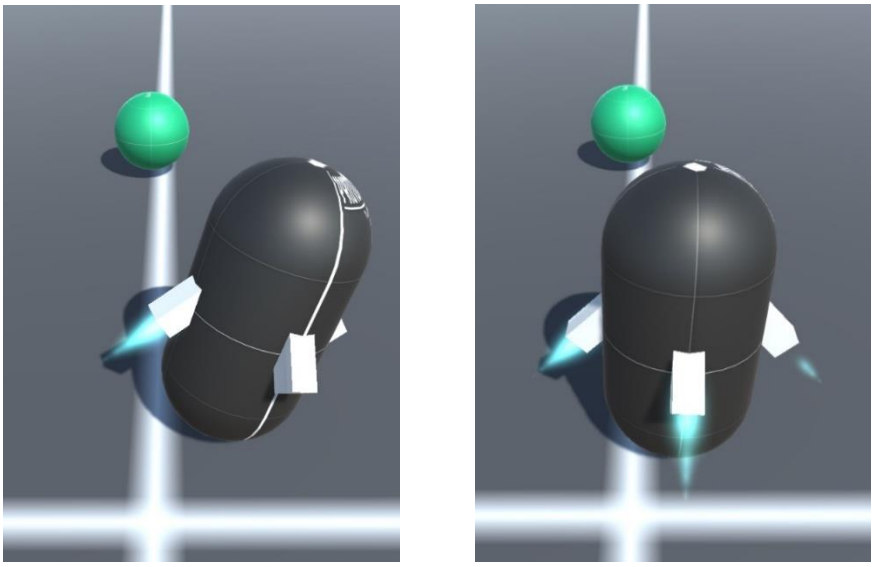


Abbildung 2-2: Screenshot Rakete

Sowohl für das KI-Training als auch für den Menschen ist es notwendig, eine Art Ziel einzuführen, welches erreicht bzw. eingesammelt werden muss. Dafür gibt es grüne Punkte in der Spielwelt, welche durch Kollision eingesammelt werden und die Punktzahl erhöhen. Diese sollen zufällig verteilt werden.

Um die Rakete sowohl für KI als auch für menschliche Eingaben einfacher zu steuern, soll sie sich wie ein Steh-Auf-Männchen verhalten. Dabei wird der Schwerpunkt nach unten verlagert, sodass das Objekt sich immer wieder von allein senkrecht aufrichtet.

3 Recherche

Vor Projektbeginn wurde zu verschiedenen Themengebieten recherchiert, die sich um das Thema der künstlichen Intelligenz drehen.

3.1 Artificial Design

Die erste Recherche innerhalb dieser Arbeit und dieses Moduls bezog sich auf *Artificial Design*. Damit ist Design gemeint, welches durch (oder mithilfe von) künstlicher Intelligenz erzeugt wird. Dies wird immer populärer, da Computer immer leistungsstärker werden und Aufgaben immer schneller lösen können. Mittlerweile kann ein Heimcomputer auch aufwändigere KI-Berechnungen durchführen.
Damit einhergehend werden die Algorithmen besser und führen zu besseren Ergebnissen, da sie nun viel mehr Rechenleistung zur Verfügung haben.

Das Konzept der computergestützten Konstruktion begann in den 1960er Jahren, als die ersten Computer programmiert wurden, um dreidimensionale Modelle zu erzeugen.
Darauf aufbauend wurden in den 1970er Jahren die ersten CAD-Systeme entwickelt, die es den Ingenieuren und Designern ermöglichten, komplexe technische Zeichnungen und Konstruktionszeichnungen zu erstellen. Diese Systeme waren jedoch sehr teuer und nicht für den Heimgebrauch geeignet.

Abbildung 3-1: 1970er: Harold Cohens AARON (Malmaschine)

Ab den 1980er Jahren war es möglich, CAD-Software auf Heimcomputern zu benutzen, was die Technologie zugänglicher machte. Die Grafik- und Prozessortechnologie verbesserte sich, was die Erstellung von realistischeren und detaillierteren dreidimensionalen Modellen ermöglichte.

Im 21. Jahrhundert hat sich das Artificial Design erneut stark weiterentwickelt. Es gibt jetzt spezialisierte Software für verschiedene Branchen wie Architektur, Maschinenbau oder Produktdesign. Ab 2014 wurden *Generative Adversarial Networks* (GANs) eingeführt, welche zu einem großen Sprung in KI-gestützter Bildgenerierung geführt haben. GANs sind eine Art künstliches neuronales Netzwerk, die zur Generierung von neuen Daten verwendet werden können. Darauf aufbauend wurden ab 2016 Text-To-Image-Anwendungen veröffentlich. Darüber können Bilder anhand von Texteingaben erzeugt werden (bspw. DALL-E).

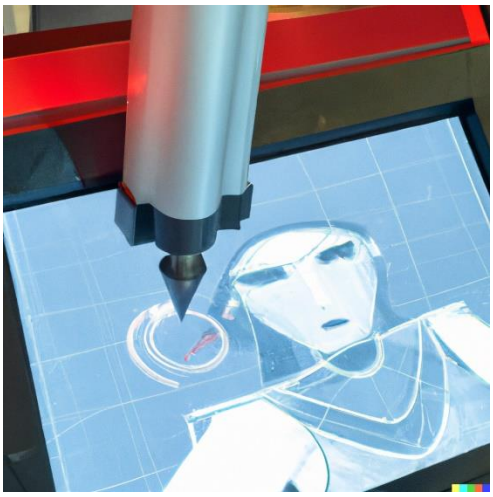


Abbildung 3-2: DALL-E-Zeichnung: „intelligent machine drawing an image“

Insgesamt hat sich das Artificial Design im Laufe der Jahrzehnte enorm weiterentwickelt, von den ersten CAD-Systemen bis hin zur heutigen Welt von *Virtual Reality*, künstlicher Intelligenz und generativem Design. Die Technologie hat die Art und Weise verändert, wie Designer und Ingenieure ihre Ideen visualisieren, entwerfen und testen, was zu einer höheren Genauigkeit, Effizienz und Produktivität in einer Vielzahl von Branchen führt.

3.2 Deep Learning

Im zweiten Teil der Recherche wurde *Deep Learning* betrachtet, da dies ein elementarer Bestandteil der zu bauenden Anwendung sein wird. Die Rakete soll mittels Deep Learning gesteuert werden.

Deep Learning ist ein Teilbereich des maschinellen Lernens, der auf *künstlichen neuronalen Netzen* basiert. Es handelt sich dabei um komplexe Modelle, die in der Lage sind, hierarchische Abstraktionen von Daten zu lernen. Im Gegensatz zum traditionellen maschinellen Lernen, das sich auf die Entwicklung von Algorithmen und Regeln konzentriert, die manuell in den Algorithmus eingefügt werden, ermöglicht Deep Learning eine automatisierte Extraktion von Merkmalen aus den Daten.

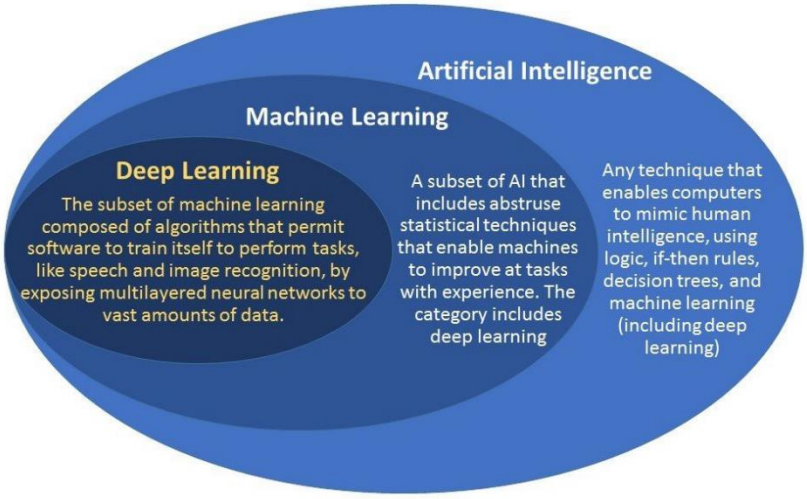


Abbildung 3-3: Einordnung von Deep Learning innerhalb des Bereichs der künstlichen Intelligenz (Artificial Intelligence)

Deep Learning basiert auf künstlichen neuronalen Netzen, die aus vielen miteinander verbundenen Schichten von Neuronen bestehen, die in der Lage sind, Daten zu verarbeiten und zu klassifizieren. Diese Schichten sind hierarchisch angeordnet und werden nacheinander angewendet, um schließlich ein endgültiges Ergebnis zu liefern.

Deep Learning wird häufig zur Verarbeitung von großen Datenmengen eingesetzt, wie beispielsweise in Bilderkennung, Spracherkennung und *Natural Language Processing* (NLP). Ein Beispiel für den Einsatz von Deep Learning ist die Erkennung von Objekten in Bildern, bei der das neuronale Netzwerk lernt, Merkmale wie Kanten, Formen und Texturen aus den Bildern zu extrahieren, um schließlich eine Vorhersage zu treffen, welches Objekt abgebildet ist.

Ein weiterer Vorteil von Deep Learning ist, dass es in der Lage ist, aus großen Datenmengen automatisch zu lernen, ohne dass menschliche Experten vorab Wissen in das Modell einpflegen müssen. Dies ermöglicht eine höhere Flexibilität und Skalierbarkeit bei der Datenverarbeitung. Deep Learning ermöglicht Maschinen, Aufgaben zu lösen, die für Menschen schwierig oder unmöglich sind. Dies ist genau das, was in diesem Projekt demonstriert werden soll.

Trotz der Vorteile von Deep Learning gibt es auch Herausforderungen, wie zum Beispiel die Notwendigkeit großer Trainingsdatensätze, die Berechnungskosten und die Schwierigkeit, die Entscheidungen eines Deep Learning-Modells zu interpretieren.

3.3 Recherche zu Umsetzungsmöglichkeiten

Vor der Projektumsetzung wurde recherchiert, welche Lösungsansätze es bereits gibt und wie künstliche Intelligenz in Unity integriert werden kann. Dabei hat sich schnell herausgestellt, dass Unitys ML-Agents der richtige Ansatz für die Aufgabenstellung ist.

Dies ist ein Framework für die KI-Erstellung) in Unity-Anwendungen. Das Framework basiert auf der Integration von PyTorch (Python). Es bietet ein komplettes Toolkit für die Entwicklung von KI-Modellen, das es Entwicklern ermöglicht, Trainingsumgebungen und -szenarien zu erstellen, Modelle zu trainieren und zu testen und schließlich die trainierten Modelle einzusetzen.

Unity ML-Agents unterstützt die Entwicklung von verschiedenen Arten von KI-Modellen, wie z.B. Reinforcement-Learning-Modelle, Supervised-Learning-Modelle und Unsupervised-Learning-Modelle.

4 Zeitplanung

Als der Rahmen des Projekts abgesteckt und die Aufgaben grob vorhanden waren, wurde eine Zeitplanung erstellt. Diese wurde in die kommenden Monate eingeteilt. Gleichzeitig sollte jeder Monat auch durch einen Meilenstein repräsentiert werden.

1. Meilenstein: Prototyp mit KI

Zeitraum: Dezember 2022

- Entwicklung Raketen-Prototyp in Unity
- Recherche zu KI
- Einbindung der KI ins Projekt
- Prãsentation des 1. Meilensteins

2. Meilenstein: KI wurde konfiguriert

Zeitraum: Januar 2023

- Weiterentwicklung der Rakete
- KI im Projekt anpassen/konfigurieren
- KI trainieren lassen
- Visualisierungen hinzufügen
- Prãsentation des 2. Meilensteins

3. Meilenstein: Projektfertigstellung

Zeitraum: Februar 2023

- Weiterentwicklung der KI
- KI trainieren lassen und ggf. Anpassungen vornehmen
- Fertigstellung des Projekts
- Fertigstellung Dokumentation
- Finale Projektpräsentation

5 Projektinitialisierung

Bevor das Projekt umgesetzt werden konnte, mussten einige Bestandteile installiert und konfiguriert werden. Als Betriebssystem wird dabei Windows 10 verwendet und die Befehle in der Kommandozeile ausgeführt.

Unity
Als Grundlage der Anwendung dient Unity in der Version 2021.3.4f1 (LTS).

Python
Das von ML-Agents verwendete Framework PyTorch ist in Python programmiert. Daher muss eine Python-Umgebung installiert werden. PyTorch benötigt in der verwendeten Version Python 3.7.9.

Nach der erfolgreichen Installation muss für Python eine virtuelle Umgebung (venv) in einem geeigneten Ordner angelegt werden:

```
py -m venv venv
```

PyTorch
In diesem Projekt wird PyTorch in der Version 1.7.1 verwendet und installiert:

```
pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html
```

ML-Agents
Das ML-Agents-Framework wird in der Version 0.29.0 verwendet und installiert:

```
pip install mlagents==0.29.0
```

Außerdem gibt es für ML-Agents ein Unity-Package, um es einbinden zu können. Dies wird über den Package-Manager hinzugefügt.

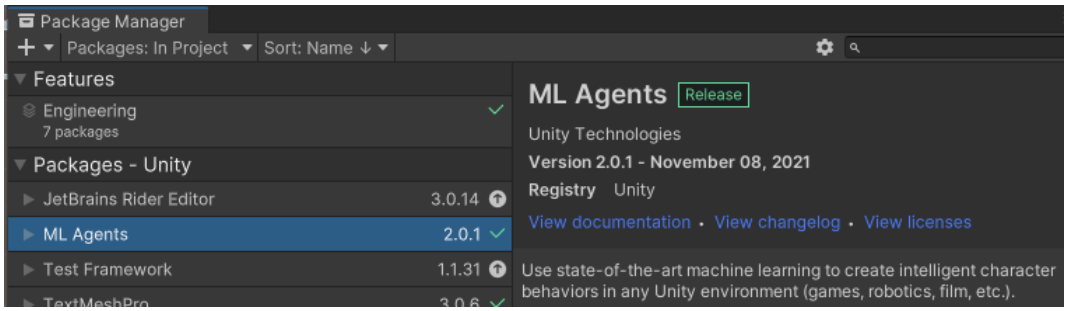


Abbildung 5-1: Unity Package Manager

Dabei ist zu beachten, dass die jeweils angeblich geforderten Versionen nicht unbedingt mit den Versionen der anderen Programme harmonieren. Beispielsweise benötigt ML-Agents in Version 0.30.0 die PyTorch-Version 1.7.1 und eine Python-Version >= 3.8.13. Jedoch unterstützt PyTorch 1.7.1 keine der Python-Versionen, die von ML-Agents 0.30.0 benötigt werden.
Daher wurde in diesem Projekt mit teilweise älteren Versionen gearbeitet.

6 Umsetzung des Projekts

Nachdem die notwendige Software aufgesetzt wurde, konnte das Projekt realisiert werden. Als grundlegende Spiellandschaft dient eine viereckige Ebene.

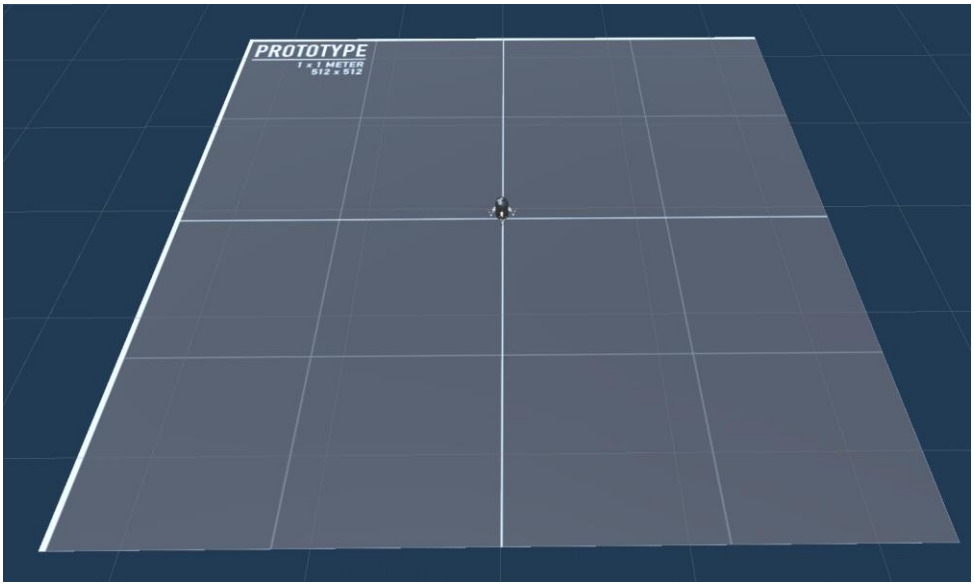


Abbildung 6-1: Spielebene

6.1 Steuerbares Objekt (Rakete)

Ein sehr wichtiger Bestandteil der Anwendung ist das raketenähnliche Objekt, welches gesteuert werden soll. Die Rakete verwendet vier Triebwerke, damit sie stabilisiert bewegt werden kann. Da immer ein Triebwerk auf der gegenüberliegenden Seite vorhanden ist, kann notfalls immer eine Gegenbewegung durchgeführt werden. Die aktive Schubdüse spielt dabei einen Partikeleffekt ab, damit erkennbar ist, ob die Eingabe funktioniert.

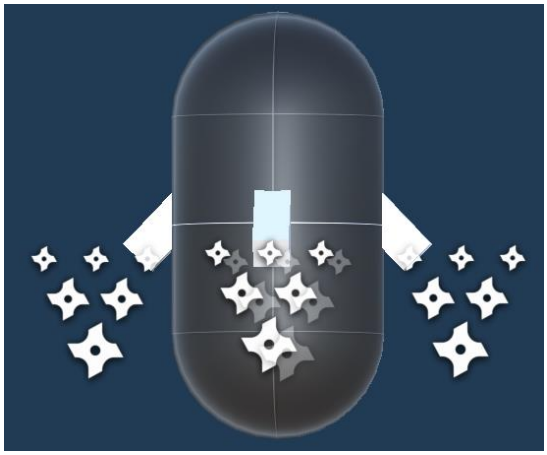


Abbildung 6-2: Rakete mit Triebwerken

Die Rakete soll sich wie ein Steh-Auf-Männchen verhalten, daher darf der Schwerpunkt nicht in der Mitte, sondern relativ weit unten liegen. Dazu wird dies am Anfang so festgelegt:

```
initialCenterOfMass = new Vector3(0, -4f, 0);
this.rb.centerOfMass = initialCenterOfMass;
```

6.2 Collectible (einsammelbarer Punkt)

In der Anwendung wird auch ein Ziel benötigt, welches die Spieler (egal ob menschlich oder KI) einsammeln können, um Punkte zu erhalten. Dafür wird ein grüner Punkt verwendet, der an einer zufälligen Position innerhalb der Spiellandschaft platziert wird.
Wenn die Spielfigur mit dem Punkt kollidiert, wird dieser an eine neue zufällige Position gesetzt. Es gibt immer nur einen Punkt. Es wurde sich gegen mehrere Punkte entschieden, um ein eindeutiges Ziel für die KI zu haben. Außerdem müssen KI und Spieler somit um den gleichen Punkt kämpfen.

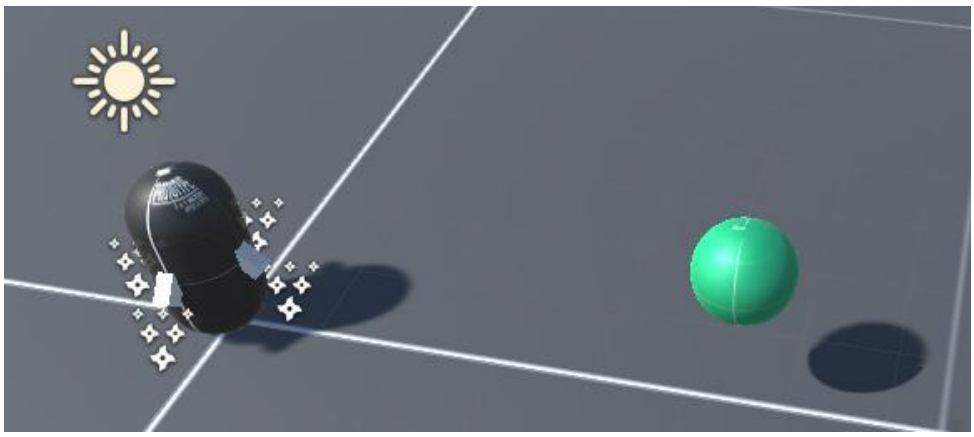


Abbildung 6-3: Einsammelbarer Punkt (Collectible)

6.3 Zurücksetzen der Spielfigur

Da die Spielebene an den Rändern keine Begrenzung hat, musste eine Lösung gefunden werden, wenn die Rakete herunterfällt. Dazu wurde an den vier Seiten, an der Decke und am Boden unterhalb der Ebene Collider angebracht. Wenn die Rakete damit kollidiert, wird sie in die Mitte zurückgesetzt.

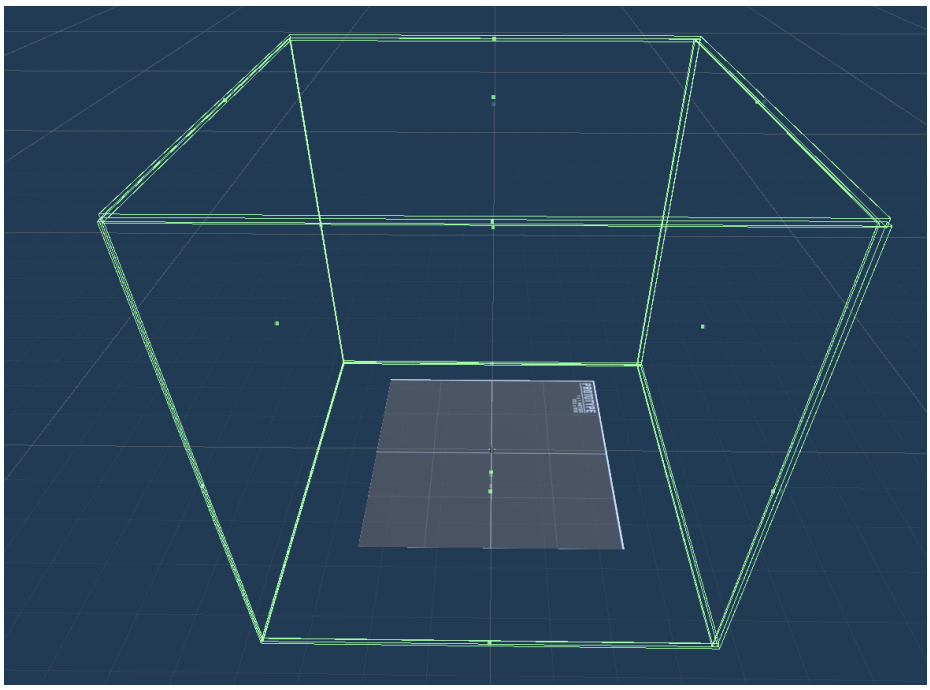


Abbildung 6-4: Collider für Spielfeldbegrenzung

Ebenso muss die Spielfigur während des Trainings zum Beginn jeder Episode (ein Trainingsdurchlauf) zurückgesetzt werden.

Es reicht jedoch nicht, nur die Position zurückzusetzen, da gleichzeitig auch die Rotation und die Drehgeschwindigkeit zurückgesetzt werden müssen. Dazu werden zu Beginn die initialen Werte zwischengespeichert und bei Kollision wieder angewendet:

```
private void ResetAgent()
{
    transform.rotation = initialRotation;
    transform.position = initialPosition;
    rb.angularVelocity = initialAngularRotation;
    rb.centerOfMass = initialCenterOfMass;
    rb.velocity = new Vector3(0, 0, 0);
}
```

6.4 KI-Anbindung an Rakete

Nach der Installation kann ML-Agents im Projekt verwendet werden. Nachdem man der Rakete ein *Agent*-Skript gegeben hat, wird das Objekt als Agent erkannt und automatisch die *Behavior Parameters*-Komponente hinzugefügt. Darüber hat man die Möglichkeit verschiedenste Parameter für das Verhalten der künstlichen Intelligenz festzulegen:

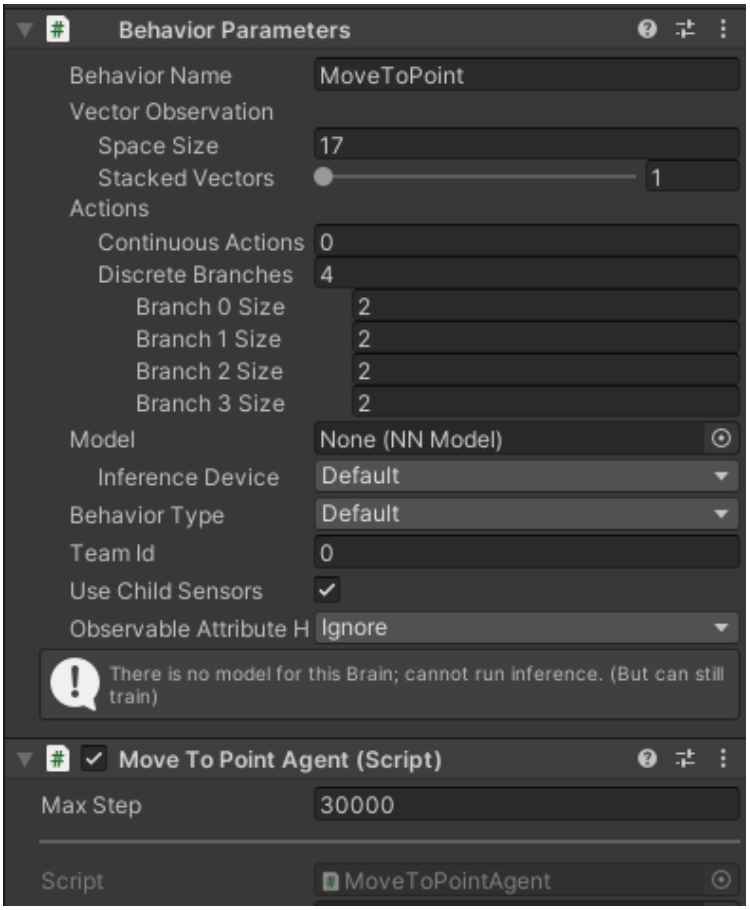


Abbildung 6-5: Agent-Verhaltensparameter

Besonders wichtig sind hierbei die *Observation* (Sensoren, die zur Laufzeit überwacht werden) und die *Discrete Actions*, die anschließend genauer betrachtet werden.

6.4.1 Sensorik

Das verwendete KI-Framework bietet die Möglichkeit verschiedene Parameter (beispielsweise Positionen) durch die Agents überwachen zu lassen (Observation). Dadurch wissen sie, wo sie sich und das Ziel (Collectible) befinden. Es wurde sich bewusst gegen maschinelles Sehen oder ähnliche Sensoren entschieden, da diese für den Anwendungsfall nicht notwendig waren. Da immer nur ein Collectible vorhanden ist, gibt es über die Positionsüberwachungen genügend Informationen.
Folgende Informationen überwacht jeder Agent:

- Lokale Position
- Drehung (jeweils für X-, Y- und Z-Achse)
- Drehgeschwindigkeit um Y-Achse (wichtig, da schnelles Drehen bestraft wird)
- Position des Collectibles (Ziel)
- Wie viele Collectibles im aktuellen Durchlauf eingesammelt wurden
- Distanz zwischen Collectible und Rakete
- In welcher Richtung das Collectible liegt

6.5 Raketensteuerung

Dieses Kapitel klingt wie Raketenwissenschaft, ist es jedoch nicht unbedingt. Wie zuvor beschrieben, besitzt die Rakete vier Schubdüsen, die an sich nur der Optik dienen. An der gleichen Position wird jedoch auch Kraft auf die Rakete ausgeübt, um sie zu bewegen.

Dabei muss Schub sowohl in die horizontale Richtung zum Bewegen produziert werden als auch in die vertikale Richtung, damit die Rakete abheben kann. Daher wird die Kraft zu gleichen Teilen in diese beiden Richtungen gegeben. Die Implementation davon wird nachfolgend mit Codebeispielen für menschliche Tastatureingaben und für die KI aufgezeigt.

6.5.1 Steuerung der Rakete als Mensch

Bei Computerspielern ist eine Steuerung über die Tasten W, A, S und D üblich, da diese wie Pfeiltasten angeordnet sind. Da die Rakete über vier ansteuerbare Schubdüsen verfügt, bieten sich diese vier Tasten zur Steuerung gut an. Jede Taste ist dabei für ein Triebwerk zuständig.

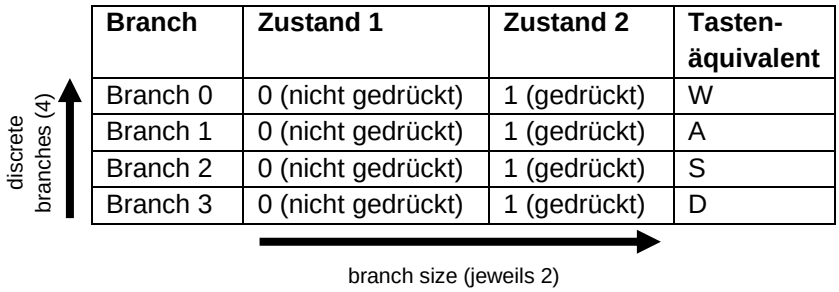
```
float thrust = 2.7f;

if (Input.GetKey(KeyCode.W))
{
    rb.AddForce(transform.forward * thrust);
    rb.AddForce(transform.up * thrust);
}
```

Beispielcode für Steuerung mittels W-Taste

6.5.2 Steuerung der Rakete als KI

Die künstliche Intelligenz in diesem Projekt arbeitet nicht mit Tasteneingaben, sondern verwendet stattdessen *Discrete Actions*, die ähnlich wie menschliche Eingaben sind.
Die Größe der D*iscrete Branches* richtet sich nach der Anzahl an unterschiedlichen Eingaben, in diesem Fall also vier ansteuerbare Triebwerke. Für jeden *Branch* wird festgelegt, wie viele unterschiedliche Aktionen auszuführen sind.



Branch	Zustand 1	Zustand 2	Tasten-äquivalent
Branch 0	0 (nicht gedrückt)	1 (gedrückt)	W
Branch 1	0 (nicht gedrückt)	1 (gedrückt)	A
Branch 2	0 (nicht gedrückt)	1 (gedrückt)	S
Branch 3	0 (nicht gedrückt)	1 (gedrückt)	D

Für den Menschen gibt es pro Taste nur zwei Zustände: entweder die Taste wird gedrückt oder sie wird nicht gedrückt. Da die KI die gleiche Steuerung verwenden soll, muss dies also auch für sie gelten. Diese zwei Zustände werden als 0 oder 1 zurückgegeben. Im Code wird die 1 als Tastendruck interpretiert. Bei einer 0 wird nichts gemacht (keine Eingabe).

```
private void MoveAgent(ActionBuffers actions)
{
    AddReward(-0.00001f);

    var w = actions.DiscreteActions[0];
    float thrust = 2.7f;

    if (w == 1)
    {
        rb.AddForce((-transform.forward) * thrust);
        rb.AddForce(transform.up * thrust);
    }
}
```

Beispielcode für Steuerung mittels W-Tastenäquivalent der künstlichen Intelligenz

Für die anderen Tasten ist der Code entsprechend ähnlich, bloß die Transform-Richtung ist eine andere.

Für jeden getätigten Schritt bekommt der Agent für die aktuelle Episode eine (sehr geringe) Strafe, damit er keine unnötigen Bewegungen durchführt. Das wird im Training näher betrachtet.

7 Training der künstlichen Intelligenz

Nachdem die Unity-Anwendung, die KI-Anbindung und die Skripte erstellt wurden, konnte mit dem Training begonnen werden.

7.1 Trainingsszenario

Das Agent-Skript musste angepasst werden, damit es für das Training sinnvoll verwendet werden kann. Diese Änderungen werden nachfolgend betrachtet.

7.1.1 Episodenbeginn

Eine Trainingseinheit besteht aus einer Episode, daher wird ein Episodenbeginn benötigt. Darin wird der Agent auf seine ursprüngliche Position (Spielfeldmitte) zurückgesetzt, damit der Ausgangspunkt immer gleich ist.

```
public override void OnEpisodeBegin()
{
    ResetAgent();
}
```

Die Funktionsweise des Zurücksetzens wurde bereits in **6.3 Zurücksetzen der Spielfigur** beschrieben.

7.1.2 Einsammeln der Punkte

Die Aufgabe in dieser Anwendung besteht darin, möglichst viele Collectibles einzusammeln, wodurch jeweils ein Punkt vergeben wird. Das Collectible besitzt einen Collider und wenn dieser mit der Rakete in Berührung kommt, wird der Punkt eingesammelt und an eine neue, zufällige Position verschoben.

7.1.3 Episodenende

Jede Trainingseinheit muss auch beendet werden. In diesem Anwendungsfall gibt es zwei Möglichkeiten zum Episodenende zu kommen:

1. **Kollidieren mit der Spielfeldbegrenzung**

Wenn die Spielfeldbegrenzung berührt wird, wird eine negative Belohnung festgesetzt und die Episode beendet.

2. **Einsammeln aller benötigten Collectibles (3 Stück)**

Wenn das Collectible drei Mal eingesammelt wurde, wird eine positive Belohnung gegeben, die Episode beendet und der Punktezähler zurückgesetzt.

7.2 Belohnungs- und Bestrafungssystem

Damit die Agents erfahren, welches Verhalten gut und welches Verhalten schlecht ist, benötigen sie Belohnungen und Bestrafungen. Dadurch wird im Laufe der Trainingszeit versucht, möglichst viel Positives zu tun. Negatives Verhalten wird abtrainiert.
Dieses System ist Bestandteil von ML-Agents und wird nachfolgend konfiguriert.

7.2.1 Altes Bewertungssystem

Ursprünglich sollte die KI immer nur einen Punkt einsammeln, danach war das Training beendet. Dies musste jedoch für spätere Trainings angepasst werden. Dafür wurde dieses Belohnungs- und Bestrafungssystem verwendet:

Art	**Reward**
Belohnung für eingesammelten Punkt	+ 1
Strafe für Berührung der Außenwand	- 1
Strafe für jeden Step	- 0,0005
Strafe für Drehgeschwindigkeit	bspw. - 0,03489

Die Strafe für Drehgeschwindigkeit war notwendig, weil die KI die Rakete sonst immer sehr schnell um die eigene Achse drehen ließ. Dies wurde so berechnet:

```
var reward = (Mathf.Abs(rb.angularVelocity.y / 100f));
AddReward(-reward);
```

Je höher die Drehgeschwindigkeit ist, desto höher fällt die Strafe aus. Das ist wichtig, damit die KI weiterhin langsame Drehungen durchführen kann, um die Rakete auszurichten. Dies konnte im nachfolgenden aktuellen Bewertungssystem ohne Anpassungen übernommen werden.

7.2.2 Aktuelles Bewertungssystem

Das alte Bewertungssystem ist für den geplanten Einsatz nicht sinnvoll, da die KI in der Anwendung viele Punkte hintereinander einsammeln soll, ohne zurückgesetzt zu werden. Das bedeutet: die KI muss nach dem Einsammeln in der Lage sein sich zu einem weiteren Punkt zu bewegen. Im alten System konnte die Rakete den beispielsweise in totaler Schieflage erreichen und das war egal, da danach die Episode beendet wurde. Um mehrere Punkte hintereinander einsammeln zu können, muss die Rakete sich immer in einer einsatzfähigen Position befinden.

Art	**Reward**
Belohnung für jeden eingesammelten Punkt	+ 1
Belohnung für Episodenabschluss (nach drei eingesammelten Punkten)	+ 2
Strafe für Berührung der Außenwand	- 1 - Anzahl eingesammelter Punkte
Strafe für jeden Step	- 0,00001
Strafe für Drehgeschwindigkeit	bspw. - 0,03489

7.3 Trainingsparallelisierung

Das KI-Training ist sehr zeitaufwändig und benötigt teilweise viele Millionen *Steps*, um auf das gewünschte Ergebnis zu kommen. Daher wurde nicht nur mit einem Agent trainiert, sondern mit mehreren gleichzeitig. Im aktuellen Stand trainieren 25 Instanzen gleichzeitig. Es wären auch noch mehr möglich, jedoch steigt die benötigte Rechenleistung dadurch weiter an.

Für die Parallelisierung war kein zusätzlicher Aufwand notwendig, die Instanzen konnten einfach dupliziert werden und haben sich dann untereinander ausgetauscht.

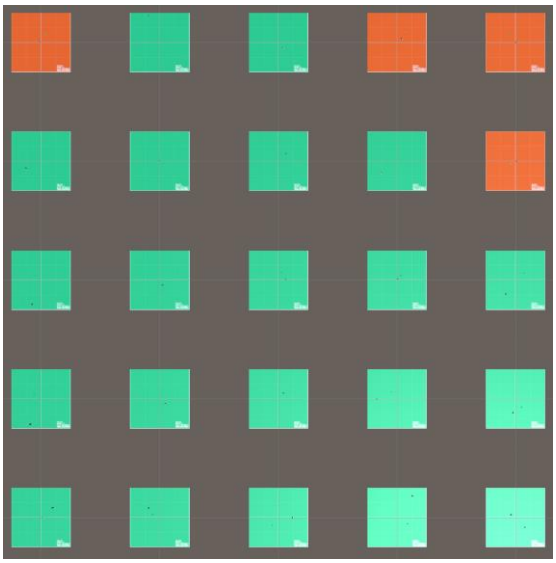


Abbildung 7-1: Paralleles Training

Gleichzeitig wurde hinzugefügt, dass das entsprechende Feld des erfolgreichen Agents grün eingefärbt wird, wenn ein Punkt eingesammelt wurde. Wenn die Rakete die Begrenzung berührt, wird das Feld orange/rot dargestellt. Dadurch ist es einfacher, den Fortschritt der 25 Instanzen zu überblicken.

7.4 KI-Lehrplan

Mit der Zeit hat sich herausgestellt, dass es nur bedingt zielführend ist, wenn die KI bis zum Ende an der gleichen Aufgabe trainiert. Wie im echten Leben kann auch der KI ein Lehrplan gegeben werden (*Curriculum Learning*). Dadurch trainiert sie die nächstschwierigere Aufgabe erst, wenn sie die vorherige Aufgabe mehrfach zufriedenstellend gelöst haben.
Über eine Konfigurationsdatei können die Lerneinheiten und das jeweilige Abschlusskriterium definiert werden.

In diesem Projekt werden die Lerneinheiten verwendet, um darüber die Höhe des Collectibles festzustellen und es nach und nach immer schwieriger zu machen.
Am einfachsten ist es, wenn das Collectible auf Bodennähe einzusammeln ist. Aus diesem Grund ist das auch der Inhalt der 1. Einheit:

```
environment_parameters:
  collectible_offset:
    curriculum:
      - name: Lesson0
        completion_criteria:
          measure: reward
          behavior: MoveToPoint
          signal_smoothing: true
          min_lesson_length: 100000
          threshold: 0.8
        value:
          sampler_type: uniform
          sampler_parameters:
            min_value: 0.1
            max_value: 1.0
```

Der *treshold* gibt an, ab welchem Durchschnittswert an erreichten Belohnungspunkten zur nächsten Lerneinheit übergegangen werden kann. Anfangs ist dieser Wert noch im stark negativen Bereich, aber nach erfolgreichem Training (wenn der Punkt zuverlässig und kontinuierlich eingesammelt wird), sollte er positiv sein.

Über das *min-* und *max_value* der *sampler_parameter* wird ein zufälliger Wert im angegebenen Bereich zurückgegeben. Dieser wird im Skript verwendet, um das Collectible zu bewegen:

```
float offset =
Academy.Instance.EnvironmentParameters.GetWithDefault("collectible_offset
", 1f);
Vector3 position = new Vector3(x, offset, z);
spawnTarget.transform.localPosition = position;
```

Nach und nach werden die Aufgaben schwieriger, also wird das Collectible immer weiter nach oben gelegt. Zum Schluss gibt es noch eine Lerneinheit, bei dem die ganze Höhe verwendet wird, damit auch niedrig gelegene Punkte weiterhin eingesammelt werden:

```
      - name: Lesson5
        value:
          sampler_type: uniform
          sampler_parameters:
            min_value: 0.1
            max_value: 10.0
```

Da dies die letzte Lerneinheit ist, werden keine Fertigstellungskriterien benötigt.

7.5 Trainingsdurchführung

Damit das Training gestartet werden kann, muss zuerst die vorher angelegte virtuelle Python-Umgebung (venv) aktiviert werden. Danach kann das Training ausgeführt werden:

Abbildung 7-2: Trainingsstart

Dabei wird der Pfad zur Konfigurationsdatei (*MoveToPoint.yaml*) mitgegeben, welches den vorher definierten Lehrplan (und andere Konfigurationen) enthält.
Anschließend muss die Anwendung in Unity gestartet werden und das Training beginnt.

7.6 TensorBoard

Im gewählten Setup wird TensorBoard automatisch mit heruntergeladen. TensorBoard ist eine leistungsstarke und interaktive Visualisierungs-Toolbox, die in das TensorFlow-Ökosystem integriert ist. Es dient zur Visualisierung, Überwachung und Fehlerbehebung von Machine-Learning-Modellen. TensorBoard bietet eine intuitive Benutzeroberfläche, die es ermöglicht, die Leistung und Genauigkeit der Modelle in Echtzeit zu visualisieren.

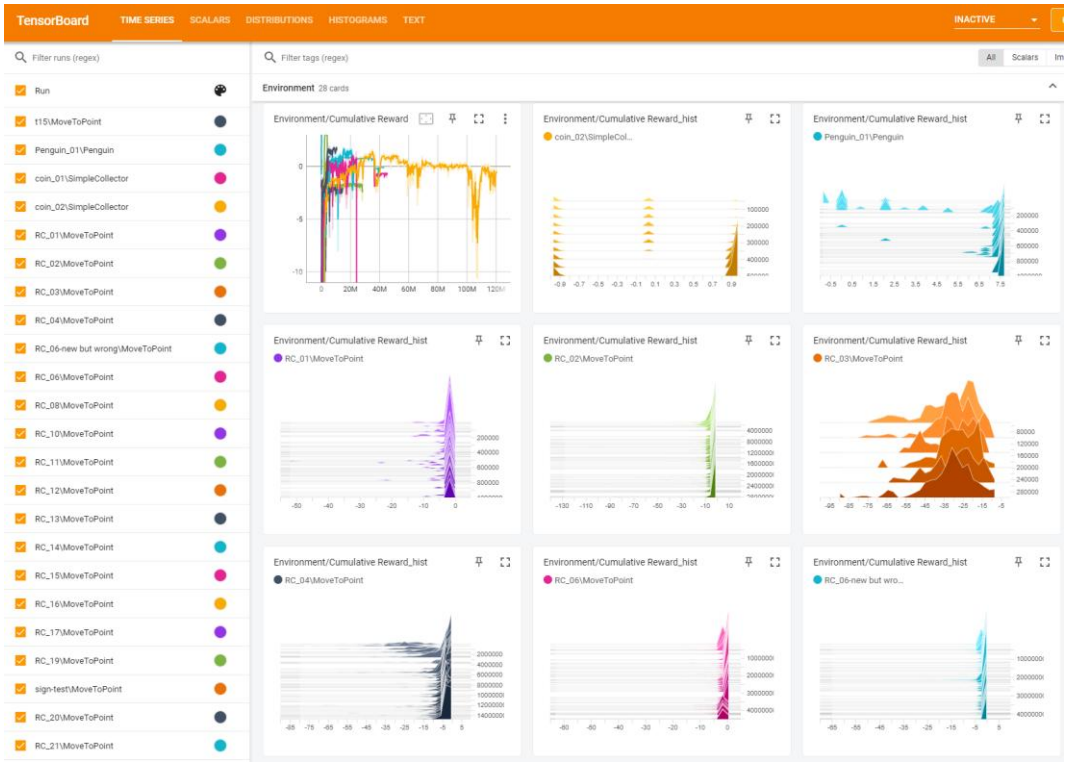


Abbildung 7-3: Übersicht TensorBoard

Während der Entwicklung war das sehr nützlich, um den KI-Fortschritt betrachten zu können. Dadurch konnte mit der Zeit auch eingeschätzt werden, ob die KI so erfolgreich lernt, wie es erwartet wird. Auch kann der Fortschritt im Lehrplan eingesehen werden.

8 Projektergebnis und -interpretation

Als Ergebnis ist eine Anwendung entstanden, in dem der Spieler (blau) gegen eine künstliche Intelligenz (schwarz) antreten kann und Punkte einsammeln kann.

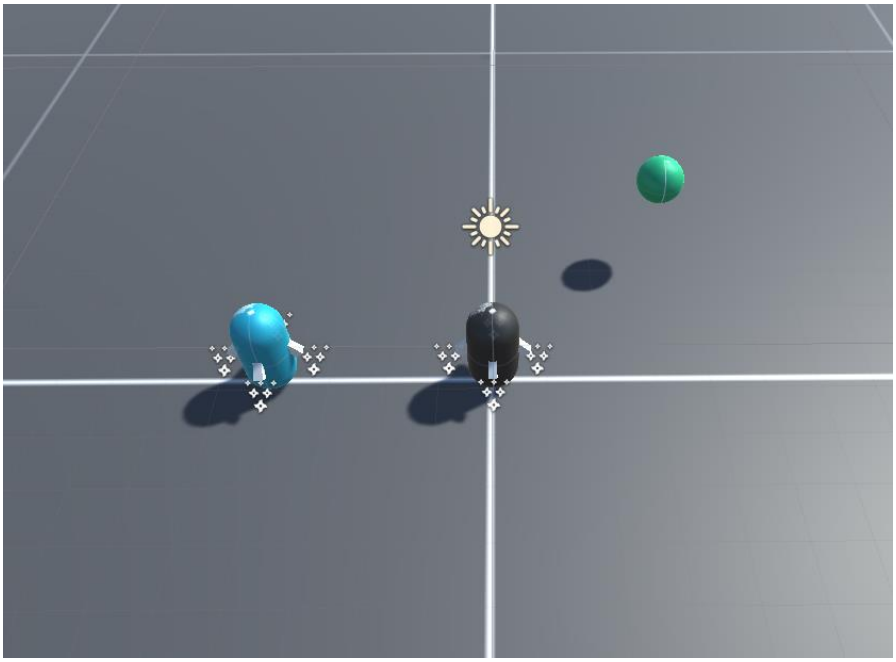


Abbildung 8-1: Spieler und KI auf dem Spielfeld

Die KI hat sehr lang mit der Anwendung trainiert. Insgesamt wurden 93 Trainings durchgeführt. Diese gingen von wenigen Minuten bis zu über 24 Stunden. Der Durchschnitt liegt bei ungefähr 2 Stunden Trainingszeit. Dabei wurden bis zu 140 Millionen Steps absolviert, der Durchschnitt liegt bei ca. 10 Millionen Steps.
Dabei sind neuronale Netzwerkmodelle entstanden, die in der Anwendung als Gehirn für den computergesteuerten Spieler verwendet werden.

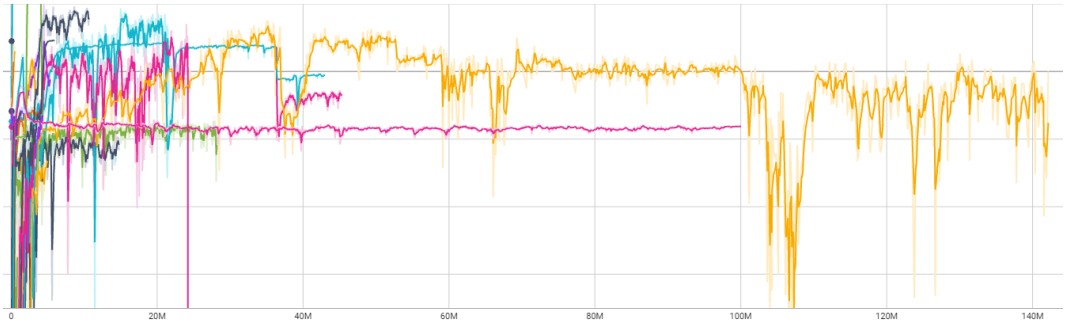


Abbildung 8-2: Steps und Reward (TensorBoard)

In der Y-Achse wird der Reward dargestellt, also mit welcher Punktzahl die Episoden beendet werden. Höher ist dabei theoretisch besser. Hierbei ist jedoch zu beachten, dass das Bewertungssystem immer wieder angepasst wurde, daher können die Durchgänge nicht immer direkt miteinander verglichen werden. Einen erfolgreichen Durchlauf erkennt man daran, dass die Linie mit der Zeit relativ gerade mit hoher Punktzahl verläuft. Das bedeutet, dass die Punkte stets gut eingesammelt werden konnten.

Dabei sind Einbrüche im Verlauf des Trainings jedoch nicht zwangsweise problematisch. Wie im Kapitel **7.4 KI-Lehrplan** beschrieben, werden im Training mehrere Lerneinheiten durchlaufen. Zu Beginn einer jeden Einheit verändert sich die erwartete Position des Collectibles und die KI muss erst lernen, wie die neuen Positionen zu erreichen sind.
Eine gut trainierte KI kann jedoch schnell mit diesem Umstand klarkommen und kehrt bald zum vorherigen Lernerfolg zurück.

Die KI zum Einsammeln eines einzelnen Punktes ist dabei sehr erfolgreich gewesen. Diese ist in der oberen Grafik in der pinken, langen Linie zu erkennen, die bis 100 Millionen Steps geht. Gerade zum Ende des Trainings verläuft die Linie einigermaßen eben. Sie kann den Punkt reproduzierbar einsammeln.

Anhand der gelben, längsten Linie erkennt man jedoch das letzte durchgeführte Training, welches mehrere Punkte einsammelt. In diesem Training war es nie möglich, konstant mehrere Punkte hintereinander einzusammeln. Das wird an einer immer wieder einbrechenden Linie ersichtlich, obwohl an diesen Stellen oftmals keine neue Lerneinheit stattgefunden hat. Meist wurde die Rakete schon vorher in die Spielfeldbegrenzung gesteuert, was zu einem vorzeitigen und damit negativen Ende der Episode führte.

Die Gründe für dieses Verhalten sind mir nicht bekannt und konnten innerhalb des Projekts trotz vieler Anpassungen nicht behoben werden. Vor allem der spontane Einbruch ab 100 Millionen Steps ist für mich unerklärlich. Es wurden keine Änderungen vorgenommen, das hätte eigentlich nicht passieren dürfen. Zu diesem Zeitpunkt hat die KI schon 19 Stunden trainiert, daher müsste bei einer Fehlerbehebung wieder so lang gewartet werden, um herauszufinden, ob der Fortschritt diesmal besser ist.

Dadurch ist die künstliche Intelligenz nicht so gut wie erhofft im Einsammeln der Collectibles. Durch diese Schwierigkeiten ist auch die Projektumsetzung verzögert worden.

9 Fazit und Ausblick

In diesem Kapitel wird ein Fazit zur Umsetzung gezogen und Herausforderungen in der Entwicklung besprochen. Im Ausblick werden Verbesserungsmöglichkeiten für die Zukunft und mögliche nächste Schritte aufgezeigt.

9.1 Fazit

Es musste zuerst ein passendes mögliches Szenario gefunden werden, welches innerhalb eines Semesters umsetzbar ist. Dabei war zu Projektbeginn eine erfolgreiche Umsetzung noch ungewiss. Da dies meine ersten Schritte mit Machine Learning in Unity sind, war der Ausgang noch ungewiss, aber die Aufgabenstellung trotzdem spannend. Selbst wenn das Projekt nicht zum gewünschten Ziel führen sollte, ist das Erlernte auf dem Weg dahin trotzdem viel wert.
Ganz nach dem Motto: der Weg ist das Ziel.

Die Umsetzung war erfolgreich, da eine künstliche Intelligenz erschaffen wurde, die die Rakete steuern und Punkte einsammeln kann.
Es gab jedoch auch viele Herausforderungen, die schon bei der Installation begonnen haben, da die jeweils angegebenen benötigten Versionen der einzelnen Komponenten nicht kompatibel waren.
Auch traten viele Herausforderungen beim Trainieren der KI auf, da das Training sehr zeitaufwändig war. Bei jedem Trainingsdurchlauf hat es teilweise mehrere Stunden gedauert, bis absehbar war, ob die KI zu einem sinnvollen Ergebnis kommen wird oder nicht.

Die KI so zu trainieren, dass sie einen einzelnen Punkt einsammelt war realisierbar. Nach mehreren Stunden Training konnte sie auch weit oben schwebende Punkte einsammeln und hat dabei nur selten Fehler gemacht.
Anders sah es jedoch aus, wenn mehrere Punkte hintereinander eingesammelt werden sollten. Das hat die KI auch nach vielen Trainingsdurchläufen und mehreren Anpassungen nicht zuverlässig geschafft. Auch der längste Durchlauf nach über 24 Stunden Training (über 100 Millionen Trainingssteps) war nur bedingt erfolgreich. Die Punkte wurden nur teilweise mehrfach hintereinander eingesammelt.

Dennoch war es spannend, den Aufbau einer künstlichen Intelligenz durchzuführen und dabei viel zu lernen. Das Curriculum Learning beispielsweise war mir vorher noch gar nicht bekannt. Daher ist das Projekt ein Erfolg für mich.

9.2 Ausblick

Bei einer Weiterentwicklung bzw. Anpassung des Projekts muss eine Lösung gefunden werden, um mehrere Punkte hintereinander zielsicher einsammeln zu können. Eventuell benötigen die Agents mehr Parameter, die sie überwachen können. Es könnte auch sein, dass die Steuerung generell zu kompliziert geplant war. Mit einem simpleren Szenario ist ein Erfolg wahrscheinlich leichter zu erzielen.
Darauf aufbauend kann die Anwendung so erweitert werden, dass der menschliche Spieler noch besser gegen die KI antreten kann. Beispielsweise könnte man eine Auswahl für die Gegnerstärke implementieren und dafür die verschieden gut trainierten neuronalen Netzwerkmodelle verwenden.