Reward-Shaping und Curriculum-Learning in einem Sparse-Reward-Adversarial-Environment in Unity3D

Luca Jedelhauser

University of Applied Sciences Hamburg
luca.jedelhauser@haw-hamburg.de

Zusammenfassung. Ziel dieser Arbeit ist die Lösung eines Sparse-Reward-Problems, mithilfe von Reward-Shaping und Curriculum-Learning, in einem Adversarial Game-Environment. Dazu wurde in Unity 3D ein 1 vs. 1 Shooter-Environment im Stil von Laser-Tag aufgebaut, in dem eine nicht-triviale Aufgabe gelöst werden muss, um einen Reward zu erhalten. Sowohl das Sparse-Reward-Problem als auch die beiden Lösungsverfahren werden in der Theorie beschrieben und dann mithilfe des Environments in der Praxis dargestellt. Anschließend folgt eine Bewertung und Gegenüberstellung der beiden Verfahren.

Schlüsselwörter: reinforcement learning · sparse rewards · curriculum learning · reward shaping

1 Einleitung

Reinforcement Learning (RL) hat besonders in den letzten Jahren für einige prominente Ergebnisse gesorgt. Sehr bekannt wurden Algorithmen die in kompetitiven Szenarien eine beeindruckende Performance erreichten. So gab es bspw. RL-Algorithmen die menschliche Experten in Brettspielen [1] oder Multiplayer Online-Games [2] schlagen konnten. Aber auch Szenarien in denen die Agenten gegen sich selbst antraten sind bekannt geworden [3][4].

Im Zuge dieser Arbeit wurde ein solches Adversarial Scenario aufgebaut, in dem zwei Agenten gegeneinander antreten. Der Aufbau orientiert sich an einem 1 vs. 1 Shooter Game, bei dem der Gegner mit einem Laserstrahl getroffen werden muss.

Da es sich hier um eine sogenannte „goal-oriented task“ handelt, bei der erst beim Erreichen eines komplexen Ziels ein Reward verteilt wird, muss das Problem der Sparse Rewards gelöst werden [5]. Dieses Problem ist sehr verbreitet im Bereich des Reinforcement Learnings und es existieren verschiedene Lösungsansätze. In dieser Arbeit sollen dafür Reward-Shaping und Curriculum-Learning herangezogen und verglichen werden. Beide Verfahren werden erst theoretisch vorgestellt und dann anhand des erstellten Unity Szenarios praktisch angewendet.

Als Ganzes ist die Arbeit in sieben Abschnitte unterteilt. Die ersten beiden beschreiben das Unity Environment und die zugehörigen RL-Environment-Aspekte. Der darauffolgende Abschnitt stellt das verwendete RL-Lernverfahren vor. Anschließend wird das Sparse-Reward-Problem kurz dargestellt und die erwähnten Lösungsverfahren theoretisch beleuchtet. Zu guter Letzt folgt die praktische Anwendung im Unity-Environment und die dazugehörige Bewertung der erhaltenen Ergebnisse.

2 Game Environment Design

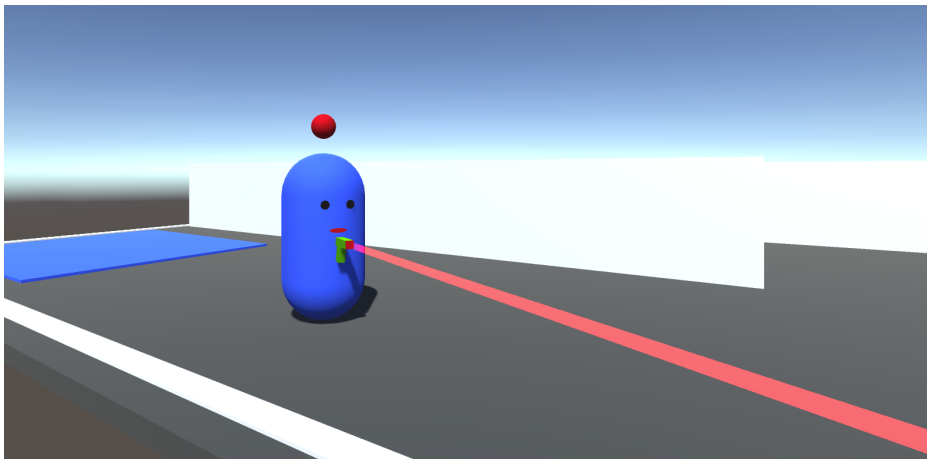

Abb. 1: Screenshot aus dem Environment in Unity 3D

2.1 Spielprinzip

Bei dem verwendeten Environment handelt es sich um ein mit Unity erstelltes Szenario, mit zwei Teams und jeweils einem Spieler pro Team. Der Aufbau orientiert sich an dem Spiel „Laser-Tag“. Die beiden Gegenspieler sind durch einen Aufbau aus Wänden getrennt, welcher durchlaufen werden muss um zur gegnerischen Seite zu kommen (siehe Abbildung 2). Gestartet wird in rechteckigen Feldern – den Startzonen. Innerhalb dieses Bereiches wird der Spieler in einer zufälligen Position und Ausrichtung in das Environment gesetzt. Beide besitzen eine Laser-Pistole, deren Strahl direkt auf den Gegner gerichtet werden muss um das Spiel zu gewinnen. Nach einem Schuss ins Leere ist der Auslöser für zwei Sekunden blockiert, bis sich die Laser-Pistole wieder aufgeladen hat. Trifft der Laser den Gegner, wird die Spielrunde beendet und beide Teilnehmer werden zurück in ihre Zonen gesetzt. Der Spieler der den Treffer erzielt hat bekommt einen Punkt. Die Wände halten den Laserstrahl auf und blockieren die Sicht der

Spieler. Die Position des Gegners ist den Spielern nicht bekannt und kann somit nur durch explorieren bestimmt werden.

2.2 Die Umgebungsvarianten

Das Environment liegt in zwei verschiedenen Varianten vor. In Variante eins muss ein Labyrinth durchlaufen werden um zur gegnerischen Seite zu gelangen. In der zweiten Variante wird das Feld in der Hälfte durch eine Wand getrennt, die links und rechts aus einem verschiebbaren Element besteht. Laufen die Spieler gegen die grünen Elemente, schieben sie diese beiseite und erreichen die gegnerische Seite. Beide Hälften bestehen zusätzlich jeweils aus einem verschiebbaren Block und zwei feststehenden Wänden. Dieses Environment soll ein komplexeres Problem darstellen, bei dem die Spieler zunächst erlernen müssen, dass bestimmte Wandabschnitte verschiebbar sind. Außerdem könnten die Elemente eventuell für intelligenteres Verhalten genutzt werden.

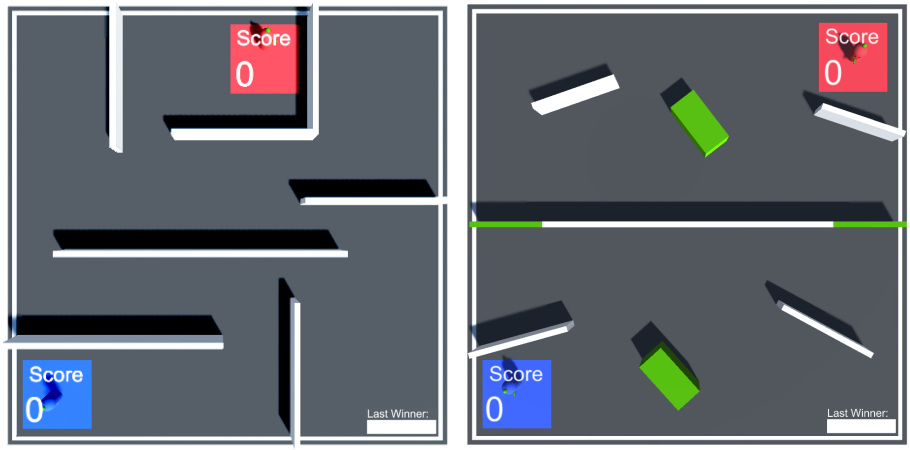

Abb. 2: Draufsicht des Spielfelds in Variante eins (links) und Variante zwei (rechts).

2.3 Steuerung

Die Spielfiguren können sich entlang zwei Achsen (X & Y) auf der horizontalen Ebene bewegen, sowie um die Z-Achse rotieren. Durch die erste Achse kann sich der Spieler vor und zurück bewegen, durch die zweite nach links und rechts. Die dritte Achse ermöglicht eine Änderung der Ausrichtung des lokalen Koordinatensystems (Blickrichtung) und damit auch ein Zielen mit der fix in X-Richtung angebrachten Laserpistole.

Die Figuren können in Unity entweder durch User-Input oder von einem Algorithmus gesteuert werden (siehe Abschnitt 4).

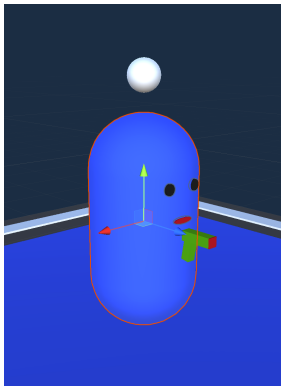

Abb. 3: Spielfigur mit lokalem Koordinatensystem. In Grün die Z-Achse, in Blau und Rot jeweils die X- und Y-Achse.

2.4 Elemente zur Visualiserung der Systemzustände

Um später gelernte Bewegungsmuster besser sichtbar zu machen, wird ein Line-Renderer eingesetzt, der bei Bedarf den Bewegungsverlauf der Spieler auf das Feld zeichnen kann.

Außerdem wird der Ladezustand der Laserpistole über eine farbige Sphäre dargestellt, die über dem Spieler schwebt. Ist die Pistole vollständig aufgeladen, ist die Sphäre grün. Betätigt man den Auslöser, bleibt sie solange rot bis die Pistole wieder schussbereit ist. So lässt sich besser nachvollziehen warum in manchen Situationen der Auslöser (nicht) betätigt wird.

Auf den Startzonen wird die aktuelle Anzahl der erfolgreichen Treffer für das jeweilige Team angezeigt.

Da in manchen Situationen nicht eindeutig erkennbar ist welches Team den Treffer gelandet hat, wird eine kleine Fläche am Spielfeldrand immer in der Farbe des Teams gefärbt, die den letzten Treffer erzielt hat. Im Reinforcement Learning Trainingsmodus wird der komplette Spielfeldboden bei einem Treffer in der jeweiligen Teamfarbe eingefärbt. Das ermöglicht auch im parallelisierten Training, wie in Abbildung 4 gezeigt, einen Überblick über den Lernfortschritt.

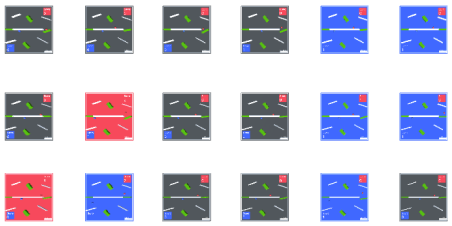

Abb. 4: Mehrere parallel trainierende Spielfelder mit gefärbtem Boden.

3 RL Environment Design

Zur Auslegung eines RL-Environments für ML-Agents in Unity gibt es in einem Open-Source-GitHub eine Reihe an „Best Practices“ [6], um ein möglichst stabilen Trainingsablauf zu erreichen. An diesen Regeln wurde sich auch bei der Auslegung dieses Environments orientiert.

3.1 Observation

Die Agenten nehmen ihre Umgebung über Ray-Casts wahr. Ähnlich einem menschlichen Spieler, ist dabei die Auflösung der Wahrnehmung im Bereich des Sichtfeldes am höchsten. Außerhalb dieses Feldes liegen die Raycasts deutlich weiter auseinander. Diese Ray-Casts ermöglichen dem Agenten aber, im Gegensatz zum menschlichen Sichtfeld, auch Hindernisse zu erkennen die sich hinter ihm befinden (der gegnerische Spieler kann außerhalb des Sichtfeldes nicht erkannt werden). Das soll dem Agenten die Möglichkeit geben, ein Verständnis davon zu erhalten, wo er sich momentan im Raum befindet. Trifft der Strahl auf ein Hindernis, wird der Abstand zum getroffenen Objekt und dessen Identifier an den Agenten übermittelt.

Zusätzlich zu den Ray-Casts kennt der Agent den Ladezustand seiner Laserpistole durch eine diskrete Input-Variable, die die zwei Zustände „geladen“ und „leer“ wiedergibt.

Seine absolute Position im Raum bekommt der Agent über Koordinaten mitgeteilt. Um eine möglichst schnelle Konvergenz zu erreichen, werden diese (wie in den Best Practices empfohlen) auf einen Bereich zwischen 0 & 1 normiert. Dabei ist aus der Sicht des Agenten die untere linke Ecke des Spielfelds der Punkt (0,0) und entsprechend die obere rechte Ecke (1,1).

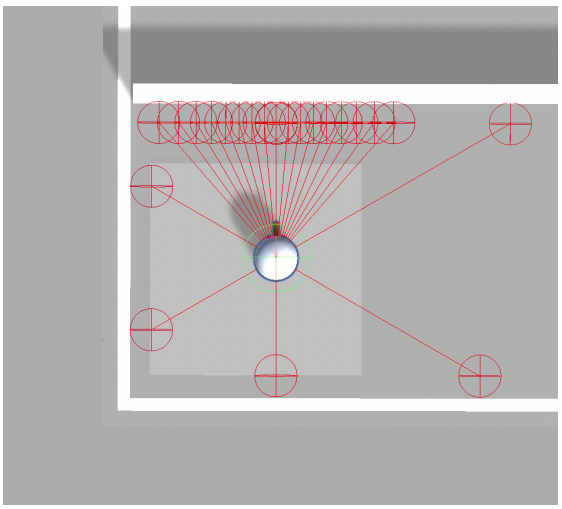

Abb. 5: Draufsicht des Spieler-Agenten mit Sensorinput.

3.2 Reward Signal

Für ein stabiles Training sollten die vom Environment vergeben Rewards den Wert 1 in der Regel nicht übersteigen [6]. Dementsprechend erhalten die Spieler für einen Treffer einen Reward von 1. Um eine Sparse Reward Situation zu erzeugen, gibt es zunächst keinerlei positiven Reward für Zwischenschritte. Um den Spieler davon abzuhalten, durchgehend den Auslöser zu betätigen, werden negative Rewards als eine Form von Munitionskosten vergeben. Pro Schuss verliert der Spieler 0.01 Reward. Der Maximal erreichbare Reward pro Runde beträgt also 0.99.

4 Das Lernverfahren

Das Lernen wird mithilfe des Open-Source-Toolkits „ML-Agents“ [7] realisiert. Das Toolkit unterstützt zwei Lernverfahren: Proximal Polixy Optimization (PPO) und Soft Actor-Critic. In der Standardkonfiguration des Toolkits wird PPO verwendet und in der Dokumentation als besonders universell und stabil beschrieben. Die Agenten des Environments lernen daher ebenfalls mithilfe von PPO. Dieses Verfahren wurde im Jahr 2017 von Schulman et al. vorgestellt [8]. Bei dem Proximal Policy Algorithmus (PPO) handelt es sich um eine Online Learning Actor-Critic Methode, aus der Kategorie der Policy-Gradient Algorithmen.

Policy-Gradient Verfahren lernen eine parametrisierte Policy $\pi(a|s,\theta)$ mit dem Parametervektor θ. Anhand einer Performancemetrik ($J(\theta)$) wird dann ein Gradient berechnet. Da diese Methoden darauf abzielen, die Performance zu maximieren, werden die Parameter über den Gradientenauftieg (Gradient Acent) nach $\theta_{t+1} = \theta_t + \alpha \Delta J(\theta_t)$ angepasst . Die Actor-Critic Verfahren lernen zusätzlich eine parametrisierte Value-Function $\hat{v}(s,w)$ mit dem Weight-Vektor w. [9]

Bei PPO wird der Gradient über einen sogenannten „Gradient-Estimator“ berechnet:

$$\hat{g} = \hat{\mathbb{E}}_t[\Delta_\theta log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{1}$$

bzw. automatisiert durch Software die folgende Policy-Gradient-Loss-Funktion differenziert:

$$L^{PG}(\theta) = \hat{\mathbb{E}}_t[log \pi_\theta(a_t|s_t)\hat{A}_t] \tag{2}$$

$\mathbb{E}_t[...]$ steht hier für einen Mittelwert über eine Reihe an Samples. Der „advantage estimator“ $\hat{A}_t$ besteht aus der Differenz der Action-Values und der Ausgaben des Critics, also der parametrisierten Value-Function und kann somit als $\hat{A}_t = Q(a_t, s_t) - v(s_t)$ beschrieben werden [10].

Führt man jedoch damit mehrere Optimierungsschritte aus, entstehen häufig zu große Policy-Updates. Daher wird im PPO-Algorithmus eine Kombination aus Einschränkungsverfahren eingesetzt, um für die Updates ein Surrogate-Loss zu berechnen, welches die Änderungen an der Policy möglichst gering hält. Die Verfahren werden in [8] genauer erläutert.

Der Ablauf eines PPO-Algorithmus ist folgend dargestellt:

Algorithm 1 PPO, Actor-Critic Style

for iteration=$1, 2, \ldots$ **do**
  for actor=$1, 2, \ldots, N$ **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimize surrogate L wrt θ, with K epochs and minibatch size $M \leq NT$
  $\theta_{\text{old}} \leftarrow \theta$
end for

Abb. 6: Ablauf des Proximal-Policy-Algorithmus. (Quelle: [8])

Jeder der N Actors sammelt parallel T Zeitschritte lang Samples mit der bisherigen Policy. Aus diesen Daten wird dann das Surrogate-Loss bestimmt und schließlich mit einem Optimizer (bspw. Adam) über K Epochen ein Optimierungsschritt ausgeführt.

4.1 Hyperparameter

Die Hyperparameter des PPO-Algorithmus wurden aus der Default-Configuration von ML-Agents übernommen. Lediglich die batch_size und buffer_size wurden geringfügig angepasst (batch_size: 128, buffer_size: 2048). Um eine gute Vergleichbarkeit zu ermöglichen, wurden die Hyperparameter über den Verlauf der Experimente nicht weiter verändert.

Die Parameter können in der ML-Agents Dokumentation [11] eingesehen werden.

5 Das Sparse-Reward-Problem

Das Problem, welches es in diesem Environment zu lösen gilt, mag für einen menschlichen Spieler zunächst trivial wirken, beinhaltet aber eine entscheidende Schwierigkeit für einen RL-Algorithmus: Das Problem der Sparse Rewards.

Von Sparse Rewards spricht man, wenn Belohnungen nur selten, bzw. erst nach komplexen Handlungsabläufen auftreten. Dadurch hat der Algorithmus Schwierigkeiten sein Handeln vor dem Auftreten des Rewards zu bewerten. Ob und durch welche Aktionen Fortschritt erreicht wird bleibt also zunächst unklar. Zusätzlich muss der Agent die Handlung aus verschiedenen Ausgangssituationen ausführen. Damit schafft auch ein einzelner, sehr selten auftretender Reward nicht automatisch Abhilfe. Wird der Reward nicht oft genug zu erreicht, kann der Agent ausschließlich explorieren und mit ziellosen Aktionsketten keinen Lernfortschritt erreichen [9].

Die Verzögerung zwischen einer Handlung und der Möglichkeit ihren Wert einschätzen zu können, nennt man auch den Reward-Horizon. Ist der Reward-Horizon zu groß um effektiv zu lernen, liegt ein Sparse-Reward-Problem vor. [12]

Der Agent in diesem Environment muss einen Parcours durchlaufen, den Gegner finden, richtig zielen und schlussendlich den Laser auslösen bevor der Gegner das tut. Wird diese Handlungskette nicht richtig ausgeführt, bekommt der Algorithmus keinerlei Rückmeldung zu seinen Aktionen.

Lässt man den Agenten in diesem Environment ohne Anpassungen lernen, zeigt sich das Problem sehr klar.

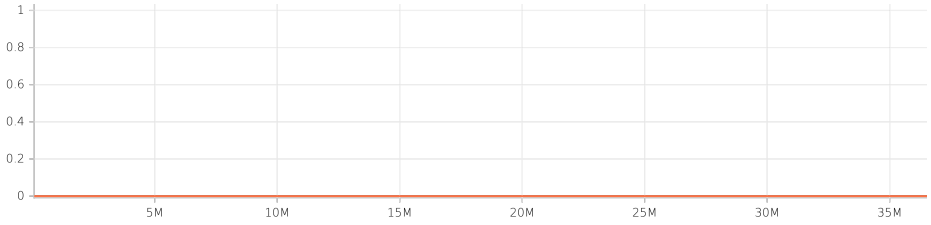

Abb. 7: Mittlerer kumulierter Reward über Millionen Episoden.

Auch nach 35 Millionen Durchläufen hat es der Agent nicht geschafft, auch nur einen Treffer zu erzielen. In der Visualisierung zeigt sich das durch ein wahlloses „Herumirren“. Auch der Laser wird zufällig ausgelöst. Es stellt sich also auch nach einer sehr hohen Anzahl an Durchläufen kein zielorientiertes Handeln ein.

Betrachtet man die Bewegung der Agenten im Environment lässt sich erkennen, dass der RL-Agent schon im ersten Schritt der Handlungskette scheitert.

Abb. 8: Bewegungsverlauf nach 35M Episoden.

6 Die Lösungsverfahren in der Theorie

6.1 Reward-Shaping

Diese Methode ist eine sehr intuitive Erweiterung des Konzepts von RL. Die Wurzeln dieses Lösungsansatzes liegen in frühen Experimenten der Psychologie. Bei der sogenannten Konditionierung können einzelne simple Handlungsschritte durch das Erzeugen einer Beziehung zwischen Aktion und Belohnung antrainiert werden [13][12]. Nach dieser Idee funktioniert auch das Reward-Shaping. Die in dieser Arbeit angewandte Variante ist das sogenannte „Subgoal-Shaping“. Dabei werden Subgoals, sprich Teilschritte, die wichtig für das Erreichen eines Ziels sind, mit bestimmten Rewards versehen. Das teilt das zu lösende Gesamtproblem in eine Reihe von kleineren Problemen auf. Der zuvor sehr große Reward-Horizon verkleinert sich auf den zeitlichen Abstand zwischen den Teilschritten. Im Kontext der Konditionierung könnte man bspw. einem Tier bereits eine Belohnung geben, sollte es sich einem Hebel nur nähern, dessen Betätigung das Ziel der Konditionierung ist. Dabei wird in der Regel Vorwissen über das Environment genutzt, um den Agenten bei der Suche nach dem optimalen Verhalten (der optimalen Policy) zu unterstützen. Die nativen Rewards des Environments werden also um synthetische „Shaping-Rewards“ erweitert. Dabei ist allerdings zu beachten, dass die Qualität dieses Feedbacks signifikanten Einfluss auf das Lernergebnis haben kann. Im Optimalfall sollen sich die optimale Policy des Prozesses mit und ohne Shaped-Rewards nicht von einander unterscheiden. Ob diese Voraussetzung beim Reward-Shaping erfüllt wurde, ist jedoch oft nicht kontrollierbar. [12]

Es ist aber klar anzumerken, dass das Handeln des RL-Agenten über die Anpassungen im Reward-Signal manipuliert bzw. gesteuert wird. Welche Zwischenschritte der Designer der Funktion als zielführend erachtet, beeinflusst also unweigerlich die daraus entstehende optimale Policy.

Die Mechanismen des Reward-Shaping stehen also in gewissem Maße im Gegensatz zu dem eigentlichen Prinzip von Reinforcement Learning. Der Agent sollte seine Lösungsstrategien selbst entwickeln, anstatt sich auf Vorwissen zu berufen, welches in einigen Anwendungsfällen auch nicht vorhanden ist.

Zu vermeidende Probleme

- Zu starker Einfluss auf das Verhalten: Wird der Agent zu stark von den synthetischen Rewards gesteuert, erhöht sich die Wahrscheinlichkeit in einem lokalen Optimum zu landen [14].
- Zyklisch erreichbare synthetische Rewards: Können die Shaping-Rewards einfach und zyklisch erreicht werden, kann der Lernalgorithmus die eigentliche Aufgabe „vergessen“ und nur noch die shaping Rewards verfolgen [15].

6.2 Curriculum-Learning

Vorgeschlagen wurde Curriculum-Learning als Unterstützung für ML-Algorithmen von Bengio et al. [16]. Die Konzepte hinter Curriculum-Learning und Reward-

Shaping liegen dabei nicht all zu weit auseinander. Die Grundidee besteht ebenfalls daraus, den Trainingsprozess zu steuern und dabei Strategien zu verwenden, die auch im Umgang mit Menschen oder Tieren funktionieren. Ähnlich dem Subgoal-Shaping werden dabei komplexe Aufgaben in kleinere „Subtasks“ aufgeteilt. Allerdings funktioniert dieser Ansatz ohne synthetische Rewards. Stattdessen wird dem lernenden Agenten nicht direkt das vollständige, komplexe Problem präsentiert, sondern eine reduzierte, einfacher zu bewältigende Version der eigentlichen Aufgabe. Aufeinander aufbauend werden die präsentierten Probleme dann zunehmend komplex. Curriculum-Learning entstand in einer Schnittmenge aus Kognitionswissenschaft und maschinellem Lernen. Dementsprechend ist diese Strategie stark deckend mit den Lernverfahren in unserem Alltag und findet sich bspw. in unserem Bildungssystem wieder. Sie ist aber auch in der Lage den Trainingsprozess neuronaler Netze zu verbessern.

Der Kern des Verfahrens besteht darin, durch eine optimierte Anordnung von simplen aufeinander aufbauenden Subtasks, die eine schnelle Konvergenz ermöglichen, die komplexe Hauptaufgabe schneller und performanter zu lösen.

Das Curriculum muss dabei aber nicht zwingend linear aufgebaut sein. Ein Subtask kann gleichermaßen auf mehrere verschiedene vorhergehende Aufgaben aufbauen. Wie genau ein solches Curriculum auszusehen hat ist generell nicht klar definiert. In der Regel wird er im Vorhinein festgelegt, kann aber auch dynamisch generiert werden. Häufig wird das Verfahren auch implizit als Teil des Trainings verwendet, ohne den Begriff als solchen zu erwähnen. Auch die Methode des Self-Play, mit der in den letzten Jahren beeindruckende Ergebnisse erzielt wurden [4][3], stellt eine Form von Curriculum-Learning dar. Es lässt sich also vermuten, dass auch in zukünftigen Errungenschaften Curriculum-Learning eine wichtige Rolle spielen wird. Für das Generieren der Subtasks gibt es verschiedene Ansätze. Beispiele sind Task-Simplification, wobei Freiheitsgrade aus der Aufgabe entfernt werden, um sie schrittweise zu vereinfachen oder Promising Initialization, einer Methode bei der die Start-States so modifiziert werden, dass sie sich näher an Rewards befinden. [17]

Zu vermeidende Probleme

- Negative transfer: Werden die Subtasks nicht sinnvoll ausgelegt, können erlernte Teilschritte der finalen Performance sogar schaden [17]. Es muss also hinterfragt werden, ob das Erlernen jedes einzelnen Teilschritts auch tatsächlich vorteilhaft für das Lösen der endgültigen Aufgabe ist.
- Falsche Anordnung der Subtasks: Ist die Reihenfolge der Subtasks nicht aufeinander aufbauend sortiert, leidet ebenfalls die Performance [17].
- Unlösbare Subtasks: Werden die Subtasks dynamisch erstellt, sollte ein Domänenexperte die Lösbarkeit der Subtasks prüfen [17].

7 Die Verfahren in der Praxis

7.1 Reward-Shaping

Um so wenig Einfluss wie möglich auf das Lernverfahren zu nehmen, soll zunächst lediglich das erste Lernproblem durch Shaping beeinflusst werden: Die Spieler verlassen ihre Seite des Spielfeldes nicht. Sie brauchen also einen Anreiz sich in die Richtung des Gegners zu bewegen. Umgesetzt wird das durch einen kleinen synthetischen Reward von 0.01, den der Agent erhält, wenn er sich einen Meter in Richtung der Mittellinie des Spielfelds bewegt. Ist ein Agent einen Meter bereits abgelaufen, kann kein weiterer Reward mehr dafür erhalten werden. So entstehen keine Reward-Zyklen. Durch die Shaping-Rewards werden die Spieler zu einem offensiveren Ansatz gedrängt. Strategien wie ein Verstecken in der eigenen Zone werden damit dementsprechend durch das Ausbleiben der Shaping-Rewards bestraft. Von einem Reward, der an den Abstand zwischen den Spielern gekoppelt ist, wurde abgesehen, da der Agent keine Informationen zum Standort des Gegners erhalten soll. Außerdem würde ein solcher Ansatz ein Zusammentreffen in der Mitte begünstigen, bei dem die Spieler trotzdem noch durch die Wand im Zentrum getrennt sind.

Erste Spielfeldvariante: Mit der ersten Spielfeldvariente zeigt sich dann, im Vergleich zum Ansatz ohne Reward-Shaping, relativ schnell ein Lernerfolg.

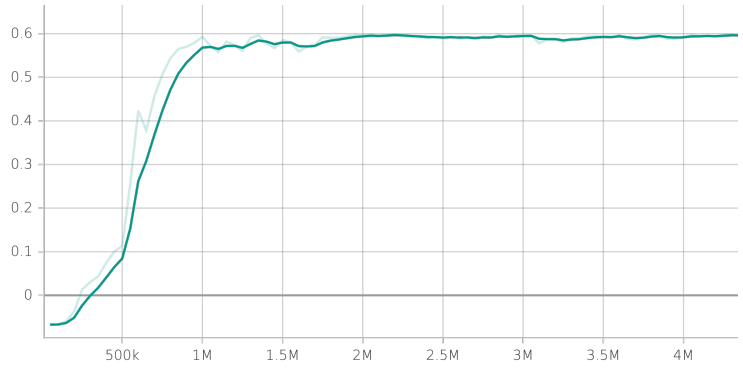

Abb. 9: Mittlerer kumulierter Reward mit Shaping über Millionen Episoden.

Schon nach etwa zwei Millionen Durchläufen scheint sich der mittlere Reward auf ein gutes stabiles Niveau eingependelt zu haben. Über die Shaped-Rewards können beide Agenten, je nach ihrer Startposition, noch maximal 0.14 Reward zusätzlich erreichen. Der durchschnittliche Reward pro Runde wird von dem ML-Agents Tool auf einen Agenten heruntergerechnet. Dementsprechend liegt der im Mittel maximal erreichbare Wert bei 0.635. In der Regel wird aber pro Spiel mehr als ein Schuss ausgelöst und die Spieler starten gelegentlich etwas

weiter in Richtung der Spielfeldmitte. Demnach pendelt sich der Wert in etwa bei 0.6 ein. Ab dieser Schwelle lässt sich über das oben gezeigte Diagramm keine Aussage mehr über die Qualität des Algorithmus treffen.

Eine weitere Möglichkeit die Fähigkeiten der Agenten zu bewerten, ist die Spieldauer. Je kürzer das Spiel, desto effizienter sind die Agenten darin, ihren Gegner zu treffen und desto weniger unnütze Handlungsschritte führen sie aus.

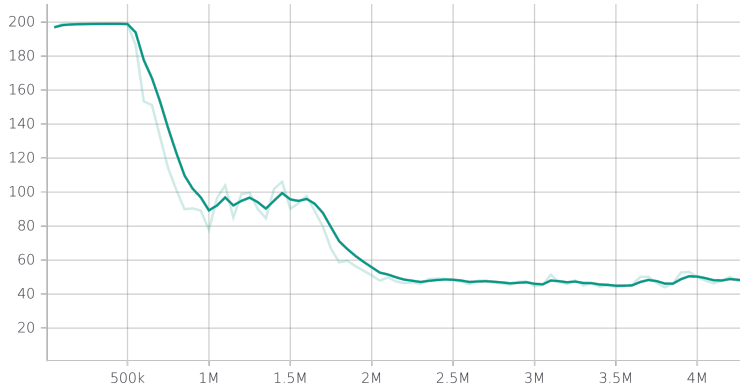

Abb. 10: Spieldauer in Render Steps über Millionen Episoden.

Die Spieldauer ist zunächst nach oben auf 200 Rendersteps begrenzt. Bis 500 Tausend Episoden läuft das Spiel gegen die Maximaldauer und wird vom Environment abgebrochen. Anschließend nimmt die Dauer jedoch stark ab und erreicht nach einer Millionen Episoden ein kleines Plateau. Nach circa 2 Millionen Durchläufen scheint sich ein stabiler Wert von circa 50 Render Steps einzustellen. Betrachtet man den Bewegungsablauf der Agenten, lässt sich eine sehr effiziente Strategie erkennen.

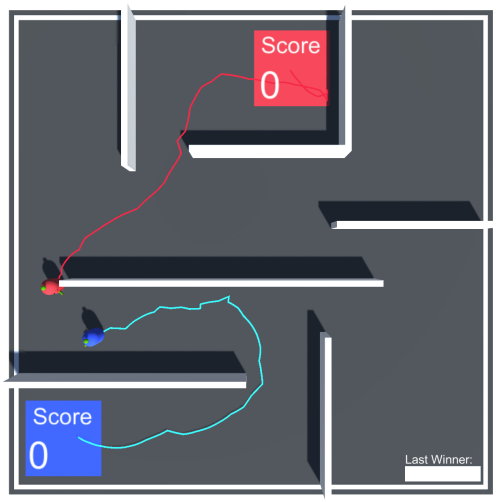

Abb. 11: Bewegungsmuster der Agenten nach vier Millionen Episoden. Die Agenten bewegen sich auf direktem Weg durch den Parcours zum Gegner.

Zweite Spielfeldvariante: Die zweite Variante bereitet dem Algorithmus größere Schwierigkeiten. Keine der begonnenen Episoden wird mit einem Treffer beendet. Lediglich die synthetischen Rewards werden erreicht. Auch mit über drei Millionen Episoden Training scheint der Algorithmus dieses lokale Optimum nicht zu verlassen.

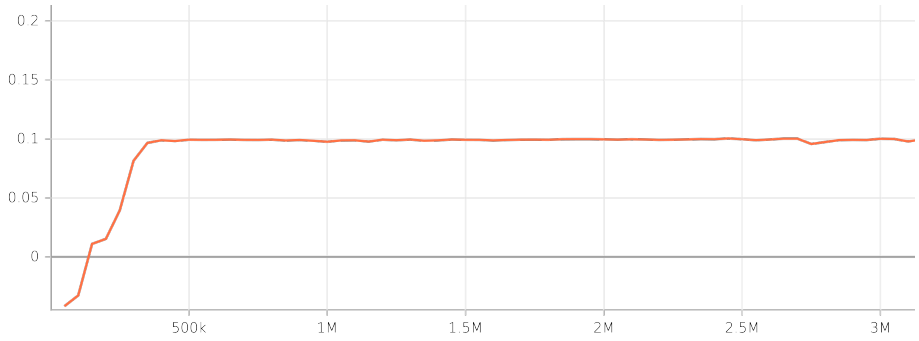

Abb. 12: Mittlerer kumulierter Reward über Millionen Episoden.

Wie sich dieses Optimum im Verhalten der Agenten widerspiegelt, lässt sich mit dem Line-Renderer gut erkennen.

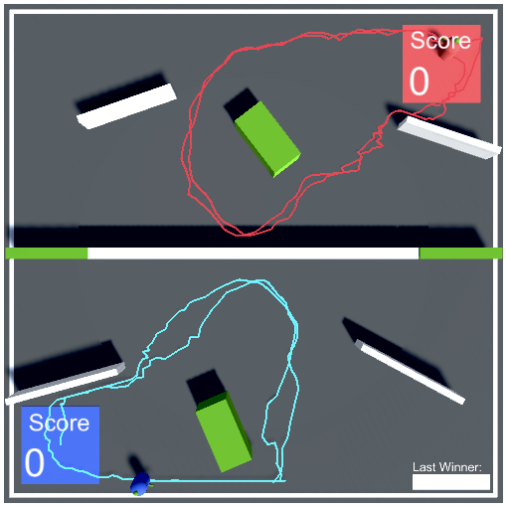

Abb. 13: Bewegungsmuster der Agenten nach drei Millionen Episoden.

Die Agenten bewegen sich aus der Startposition direkt hin zur Spielfeldmitte, drehen dann um und wiederholen ihr Bewegungsmuster. Die beweglichen Elemente der Umgebung werden nicht berührt. Um diese Variante mit Reward-Shaping zu lösen, muss also ein weiterer Anreiz geschaffen werden die Mittellinie zu durchbrechen.

Der naheliegenste Ansatz ist eine Erweiterung der synthetischen Rewards über die Mittellinie hinaus.

Möglich wäre auch ein Reward für die Annäherung an die bewegliche Wand. Allerdings stellen sich dann weitere Fragen. Es müsste bspw. geklärt werden, welche der beiden Wände angesteuert wird, und ob sich beide Agenten auf denselben Durchgang zubewegen sollen. Diese strategischen Fragen sollten idealerweise vom Lernalgorithmus herausgefunden werden. Hier werden die Grenzen von Reward-Shaping schnell deutlich. Der Lernfortschritt lässt sich nur schwer verbessern, ohne die Strategie der Agenten zu stark zu beeinflussen.

Umgesetzt wird die Erweiterung des Bereichs der synthetischen Rewards über die Mittellinie. Tatsächlich erlernen die Agenten so das Durchbrechen der beweglichen Elemente in der Wand. Allerdings liegt die Dauer für einen Spieldurchlauf deutlich über der aus Variante eins.

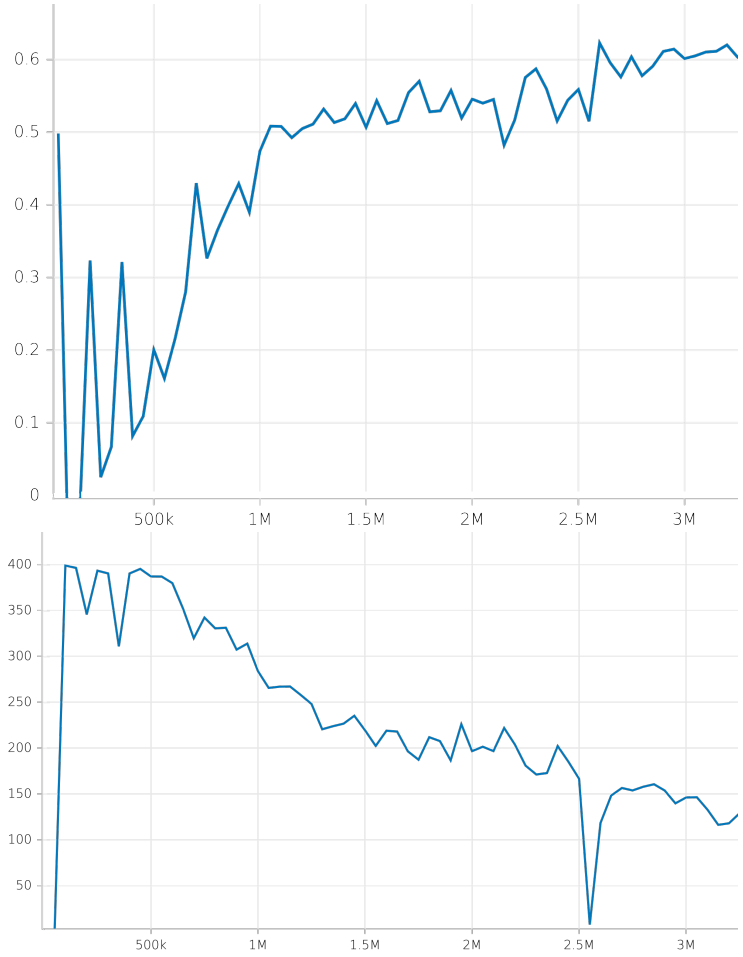

Abb. 14: Mittlerer Reward (oben) und Spieldauer in Render Steps (unten). Bei dem Ausreißer in der Spieldauer nach 2.5M Episoden handelt es sich um einen Fehler in der Berechnung. Das Training musste aufgrund eines Programmfehlers unterbrochen und wieder gestartet werden. Zum Zeitpunkt der Unterbrechung kam es bei dieser Episode zu einer Falschberechnung der Dauer.

Da es sich in Variante zwei um ein komplexeres Problem handelt, wurde die zulässige Maximaldauer für eine Episode auf 400 Render Steps erhöht. Nach über 2M Episoden liegt diese aber immer noch im Bereich von 200. Der Grund für die relativ hohe Spieldauer lässt sich wieder in der Visualisierung analysieren.

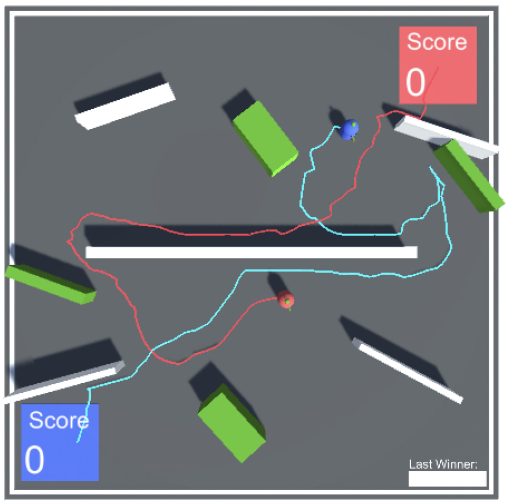

Abb. 15: Bewegungsmuster der Agenten nach 3M Episoden.

Beide Agenten haben aufgrund der synthetischen Rewards gelernt, auf direktem Weg ihre Spielfeldhälfte zu verlassen. Dadurch befinden sie sich jedoch wieder auf entgegengesetzten Hälften und müssen erneut zueinander finden. Das macht das Problem von Reward-Shaping deutlich. Die manipulierte Reward-Funktion begünstigt hier ein ineffizentes Verhalten.

7.2 Curriculum-Learning

Zur Umsetzung von Curriculum-Learning mit Unity und ML-Agents gibt es einen Abschnitt in der ML-Agents-Dokumentation [18]. Dort wird erläutert, wie sich die Kriterien für die Transitions zwischen den Subtasks setzen lassen und weitere Parameter beschrieben, die eine präzisere Einflussnahme auf den Trainingsablauf möglich machen.

Für diesen Anwendungsfall bietet sich eine Form von Task-Simplification an. Eine naheliegende Methode ist das Vereinfachen des Parcours. Durch sukzessives Hinzufügen der Hinderniswände steigt die Komplexität der Aufgabe. Die Subtasks werden jeweils in den folgenden Abschnitten der Spielfeldavarianten genauer erläutert.

Die setzbaren Hyperparameter für Curriculum-Learning in Unity sind in der folgenden Tabelle beschrieben.

Parameter	Beschreibung
measure	Bewertungskriterium für den Lernprozess
threshold	Schwellenwert für den Wechsel zur nächsten Subtask
min-lesson-length	Mindestanzahl an Durchläufen bevor zum nächsten Task gewechselt wird
signal-smoothing	Gewichtet aktuelle (0.75) mit alten Werten (0.25) wenn aktiv

Bis auf den Threshold schienen alle Parameter variantenübergreifend ähnliche Auswirkungen auf das Training zu haben. Der Threshold wurde aufsteigend, je nach Anzahl der Subtasks gestaffelt. Für den x-ten Subtask wurde aber immer derselbe Threshold verwendet. Der Unterschied bzgl. des Thresholds ergibt sich also nur durch die unterschiedliche Anzahl der Tasks. Im Sinne der Vergleichbarkeit wurden die Parameter nicht einzeln optimiert. Die besten Ergebnisse wurden, über die Varianten hinweg, mit den folgenden Werten erzielt:

Parameter	Gesetzter Wert
measure	"reward"
min-lesson-length	1000
signal-smoothing	True
threshold	0.35, 0.4, 0.46, 0.46

Erste Spielfeldvariante: Wie zu Beginn des Abschnitts erwähnt, wurde zur Realisierung von Task-Simplification ein schrittweises Hinzufügen der Wände des Parcours als Curriculum verwendet. Die einzelnen Subtasks sind in Abbildung 16 dargestellt. Die Aufteilung und Reihenfolge der Tasks wurde manuell durchgeführt.

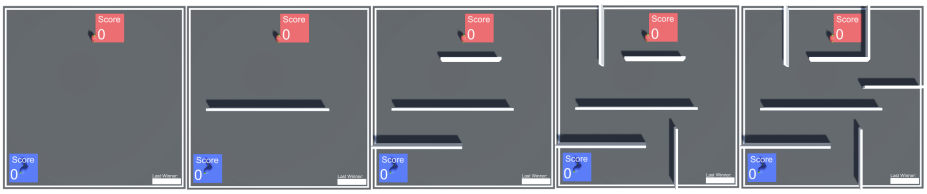

Abb. 16: Curriculum-Schritte für Spielfeld eins.

Mit dieser Konfiguration ließ sich eine schnelle Konvergenz erzeugen, jedoch überstieg die Performance nicht die des Reward-Shapings.

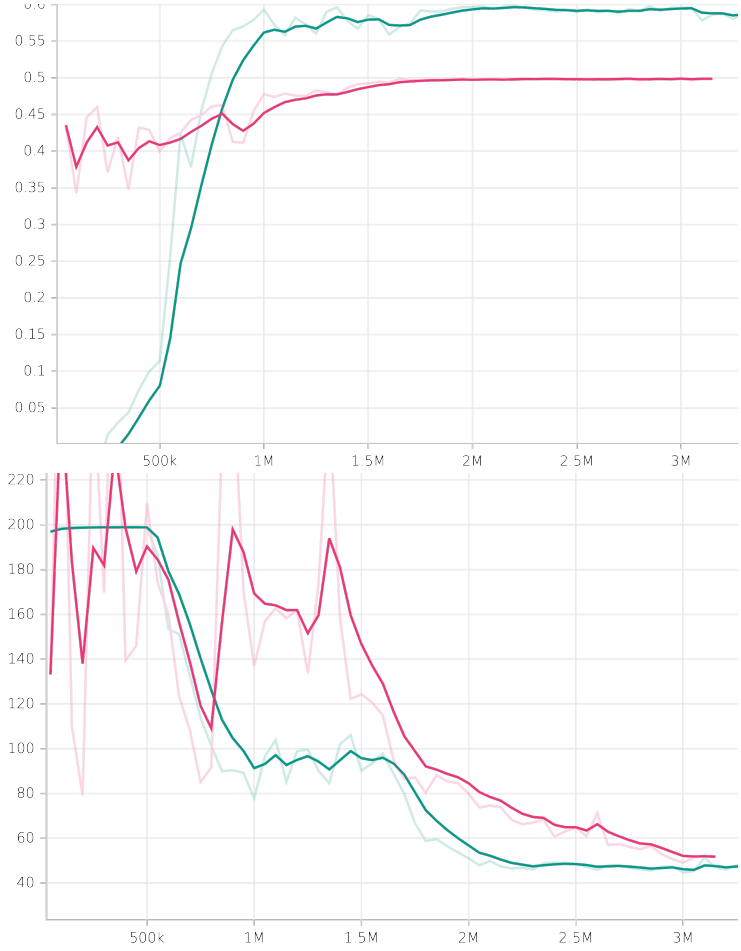

Abb. 17: Mittlerer Reward (oben) und die Spieldauer in Rendersteps (unten). Der Durchlauf mit Reward-Shaping als Vergleich in Grün.

Bezüglich des mittleren Rewards scheinen beide Verfahren ein sehr ähnliches Verhalten aufzuweisen, wobei Curriculum-Learning über die einfachen Start-Lessons logischerweise für einen größeren Reward in der Anfangsphase sorgt. Die Konvergenz hin zum maximal erreichbaren Reward stellt sich aber bei beiden nach vergleichbarer Zeit ein. Allerdings benötigt der Algorithmus mit Curriculum-Learning ca. eine Millionen Episoden mehr um auf eine ähnliche Spieldauer zu kommen. Selbst nach Optimierungsversuchen durch Änderungen in der Task-Reihenfolge und den Curriculum Hyperparametern, lies sich die Performance des Reward-Shaping nicht erreichen.

Zweite Spielfeldvariante: In dieser Variante wurden ebenfalls wieder schrittweise Wände hinzugefügt. In den ersten Versuchen dazu wurde zunächst das blanke Spielfeld verwendet und anschließend, wie in Abbildung 20 gezeigt, in vier Schritten, durch zusätzliche Hindernisse die Komplexität erhöht.

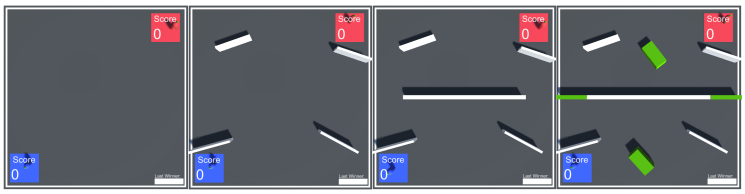

Abb. 18: Curriculum-Schritte für Spielfeld zwei

Dabei zeigte sich jedoch ein relativ instabiler Trainingsverlauf. Besonders nach dem zweiten Taskwechsel scheint der Algorithmus große Probleme zu haben, wieder eine gute Performance aufzubauen. Nach dem Wechsel bricht der mittlere Reward stark ein und benötigt ca. 750k Episoden um wieder den Ausgangszustand zu erreichen. In dieser Zeit findet auch kein Subtask-Wechsel mehr statt.

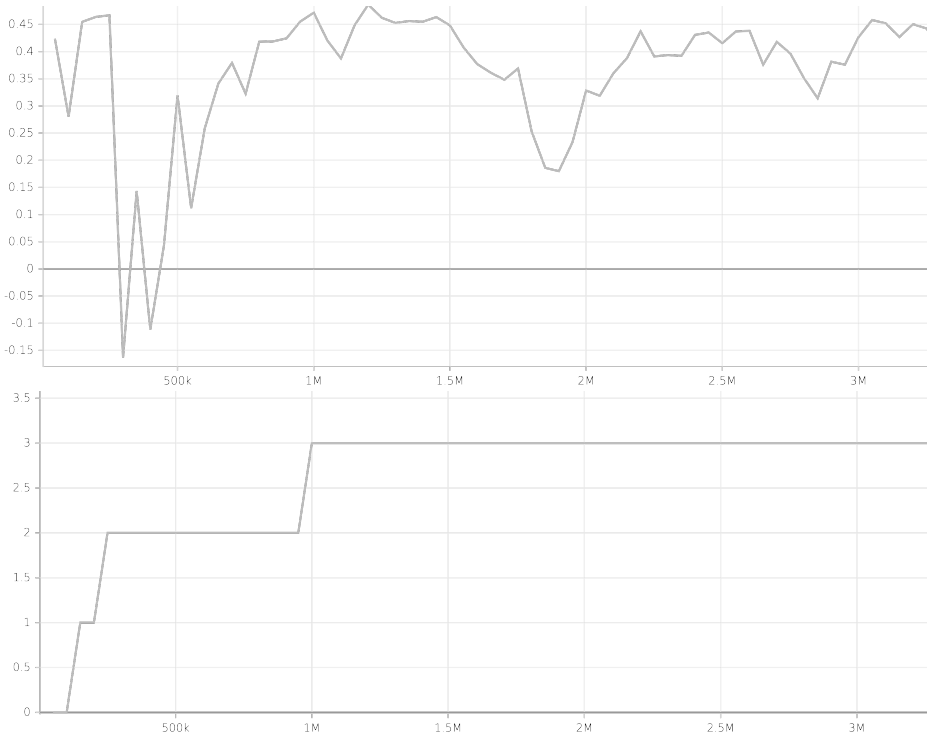

Abb. 19: Mittlerer Reward (oben) und die Anzahl der Taskwechsel des Curriculums (unten).

Um diesem Problem entgegenzuwirken, wird ein Zwischenschritt eingerichtet. Die Wand in der Spielfeldmitte wird zunächst mit reduzierter Breite hinzugefügt.

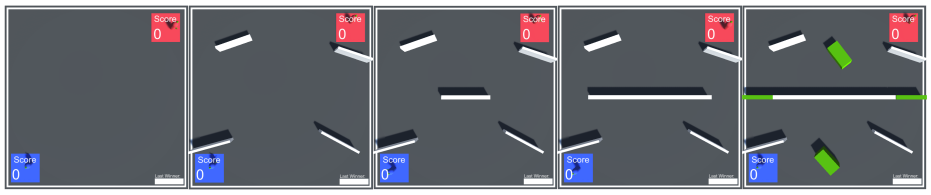

Abb. 20: Curriculum-Schritte für Spielfeld zwei mit zusätzlichem Zwischenschritt.

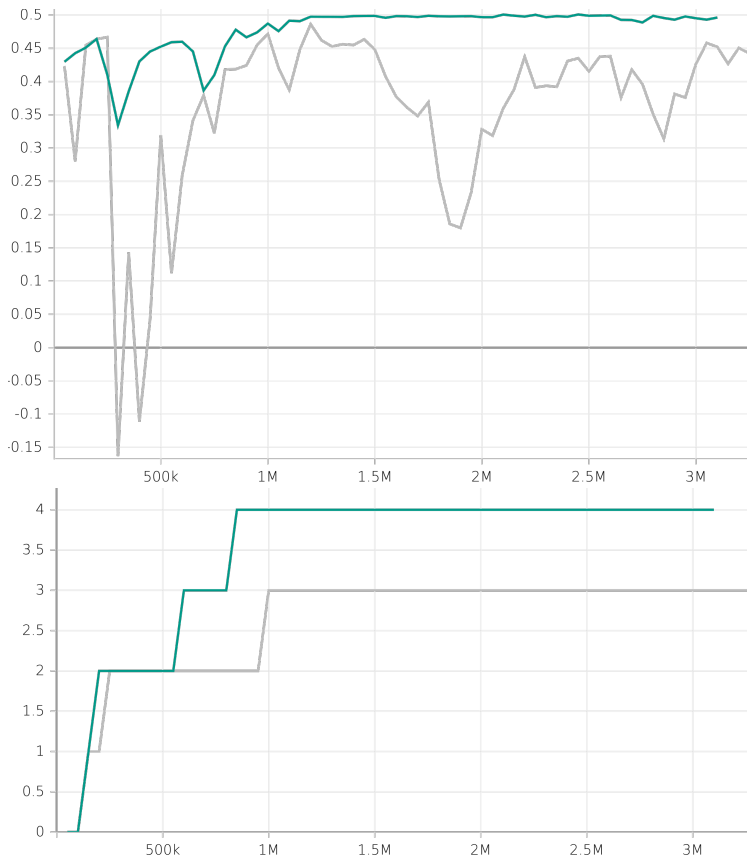

Abb. 21: Mittlerer Reward (oben) und die Anzahl der Taskwechsel des Curriculums (unten). Der Verlauf mit zusätzlichem Zwischenschritt in Grün.

Der resultierende Trainingsverlauf ist deutlich stabiler und die Einbrüche werden stark abgedämpft. Nach 1M Episoden hat der Algorithmus mit Curriculum-Learning bereits ein gutes stabiles Reward-Niveau erreicht. Auch die Episodenlänge pendelt sich nach ca. 1.5M Episoden auf einen Wert von 80 Rendersteps ein und liegt damit deutlich unter dem Ergebnis des Trainings ohne Zwischenschritt.

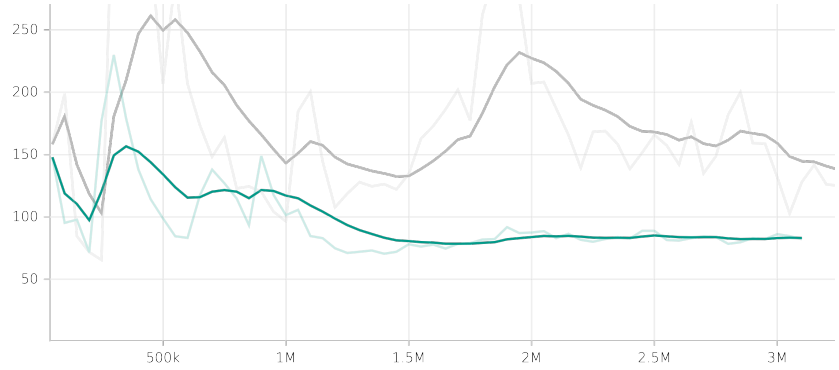

Abb. 22: Spieldauer in Render Steps über Millionen Episoden. Der Durchlauf mit zusätzlichem Zwischenschritt in Grün.

Vergleicht man die Graphen mit denen aus dem Reward-Shaping, zeigt sich klar eine verbesserte Performance durch Curriculum-Learning.

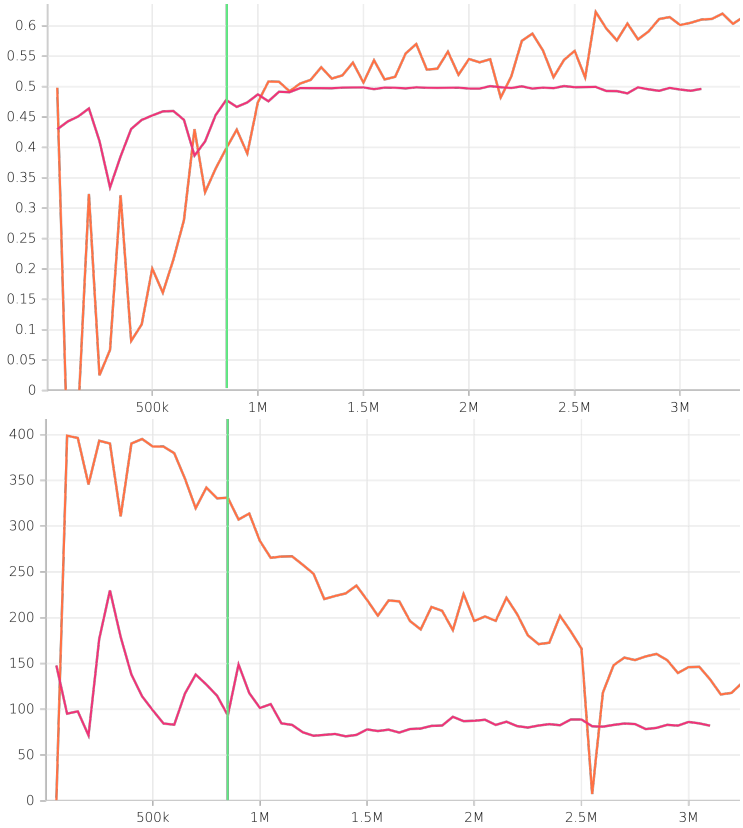

Abb. 23: Performancevergleich Reward-Shaping und Curriculum-Learning. Mittlerer Reward (oben) und die Spieldauer in Render Steps (unten). Curriculum-Learning mit zusätzlichem Zwischenschritt in Rot, Reward-Shaping in Orange. Der letzte Subtask-Wechsel des Curriculums ist bei 850k Episoden grün markiert.

Sowohl der kumulierte mittlerer Reward, als auch die Spieldauer weisen eine stark verbesserte Konvergenz auf. Besonders interessant sind jedoch die Verhaltensmuster die sich über den Trainingsverlauf von Curriculum-Learning ergeben. Da das Verfahren, im Vergleich zum Reward-Shaping, keine Handlungsschritte vorgibt, kommt es zur Entwicklung von emergenten Mustern.

Nach etwas weniger als 1M Episoden zeigt sich die folgende Strategie:

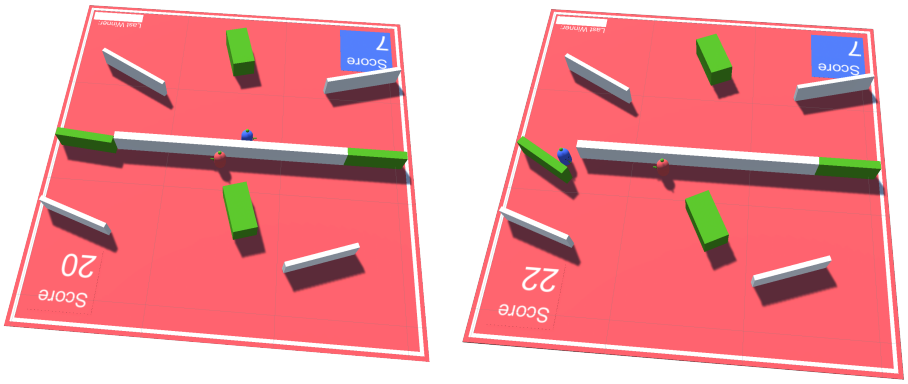

Abb. 24: Emergentes Verhalten nach 900k Episoden.

Die Agenten lauern zunächst in der Mitte des Spielfelds und „bewachen“ einen der beiden Durchgänge. Einer der Agenten durchbricht schließlich eine Barriere und es kommt zum Schusswechsel. Dieses Verhalten ist aber noch relativ ineffizient, da beide zu Beginn warten.

Ein weiteres Verhaltensmuster zeigt sich nach 2.5M Episoden. Hier bewegt sich einer der Agenten auf direktem Weg zu einer der Barrieren und stößt sie einen Spalt weit auf. Anschließend nutzt er sie als Schießscharte, um den Gegner zu treffen, ohne ihm eine zu große Angriffsfläche zu bieten.

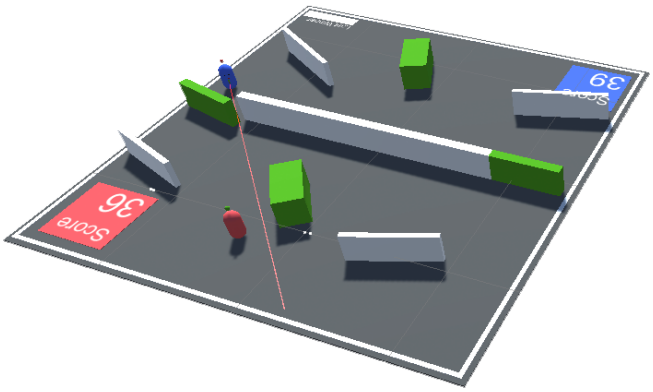

Abb. 25: Emergentes Verhalten nach 2.5M Episoden.

8 Fazit

Die Experimente konnten klar zeigen, dass Curriculum-Learning besonders für komplexe Probleme ein sinnvolles Werkzeug ist. Durch dieses Verfahren ließ sich der Trainingsverlauf stabilisieren und ein Sparse-Reward-Szenario lösen, welches im Ausgangszustand so nicht lösbar war. Bei der einfachen Umgebung brachte ein Curriculum jedoch keinen sichtbaren Vorteil im Vergleich zum Reward-Shaping. Der Algorithmus mit Reward-Shaping konnte das Problem, bezüglich der Spieldauer, sogar etwas effizienter lösen, obwohl eine geringe Spieldauer nicht Teil des Reward-Shapings war.

Beide Verfahren waren in der Lage das Sparse-Reward-Problem zu lösen. Jedoch zeigte Curriculum-Learning für die komplexere Variante des Spielfelds klar eine bessere Performance. Ein weiterer großer Vorteil des Verfahrens ist die geringere Beeinflussung der entstehenden Verhaltensmuster. So konnten sich mit Curriculum-Learning besser emergente Muster entwickeln.

Literatur

1. D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot *et al.*, "Mastering the game of go with deep neural networks and tree search," *nature*, vol. 529, no. 7587, pp. 484–489, 2016.
2. C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Dennison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse *et al.*, "Dota 2 with large scale deep reinforcement learning," *arXiv preprint arXiv:1912.06680*, 2019.
3. B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019.
4. D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton *et al.*, "Mastering the game of go without human knowledge," *nature*, vol. 550, no. 7676, pp. 354–359, 2017.
5. C. Florensa, D. Held, M. Wulfmeier, M. Zhang, and P. Abbeel, "Reverse curriculum generation for reinforcement learning," in *Conference on robot learning*. PMLR, 2017, pp. 482–495.
6. "Environment design best practices," 03 2018. [Online]. Available: https://github.com/miyamotok0105/unity-ml-agents/blob/master/docs/Learning-Environment-Best-Practices.md
7. "Github unity technologies." [Online]. Available: https://github.com/Unity-Technologies/ml-agents
8. J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal policy optimization algorithms," *arXiv preprint arXiv:1707.06347*, 2017.
9. R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.
10. J. Schulman, P. Moritz, S. Levine, M. Jordan, and P. Abbeel, "High-dimensional continuous control using generalized advantage estimation," *arXiv preprint arXiv:1506.02438*, 2015.
11. Unity-Technologies, "Training configuration file." [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-Configuration-File.md#common-trainer-configurations

12. A. D. Laud, *Theory and application of reward shaping in reinforcement learning*. University of Illinois at Urbana-Champaign, 2004.
13. A. Dickinson, "Instrumental conditioning," *Animal learning and cognition*, pp. 45–79, 1994.
14. J. Randløv and P. Alstrøm, "Learning to drive a bicycle using reinforcement learning and shaping." in *ICML*, vol. 98. Citeseer, 1998, pp. 463–471.
15. O. Marom and B. Rosman, "Belief reward shaping in reinforcement learning," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 32, no. 1, 2018.
16. Y. Bengio, J. Louradour, R. Collobert, and J. Weston, "Curriculum learning," in *Proceedings of the 26th annual international conference on machine learning*, 2009, pp. 41–48.
17. S. Narvekar, B. Peng, M. Leonetti, J. Sinapov, M. E. Taylor, and P. Stone, "Curriculum learning for reinforcement learning domains: A framework and survey," *arXiv preprint arXiv:2003.04960*, 2020.
18. "Training ml-agents," 01 2022. [Online]. Available: https://github.com/Unity-Technologies/ml-agents/blob/main/docs/Training-ML-Agents.md#curriculum