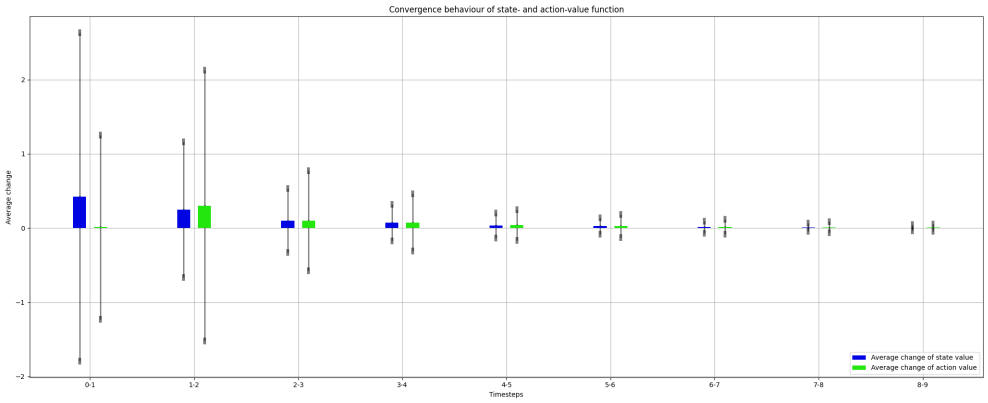

Abbildung 1: Konvergenzverhalten der State-Value Funktion und Action-Value Funktion mit zufälliger Policy.

Aufgabe 1

Aufgabe 1a)

Das Konvergenzverhalten des Algorithmus kann der Abbildung 1 entnommen werden. Dabei ist zu beobachten, dass die State-Values innerhalb weniger Zeitschritte bereits im Durchschnitt konvergieren und auch die Standardabweichung stark absinkt.

Das Konvergenzverhalten wird durch zwei Faktoren maßgeblich beeinflusst, den Diskountfaktor und die Policy unter der der Agent handelt.

Der Diskountfaktor von $\gamma = 0.9$ sorgt dafür, dass die Güte des Zustands auch von den Zuständen abhängig ist, die von ihm aus erreicht werden können. Dies hat zur Folge, dass die State-Values im Vergleich zu $\gamma = 0$ niedriger ausfallen können, da die Möglichkeit besteht, das Gitter verlassen zu wollen, was mit einem negativen Reward bestraft wird.

Die zufällige Policy erklärt vor allem die große Spannweite an State-Values die zu Anfang des Prozesses möglich sind. An zehn Stellen im Gitter ist es möglich zu 25% eine Bestrafung zu erhalten, an vier Stellen sogar zu 50%. Nur an zwei Stellen ist initial ein hoher positiver Reward zu erhalten. Da die folgenden Rewards durch den Diskountfaktor immer weniger in die Summe mit einbezogen werden verändern sich die State-Values immer weniger und konvergieren schließlich nach kurzer Zeit. Durch die hohe Wahrscheinlichkeit negative Rewards zu erhalten sind auch negative State-Values möglich. Dies wäre in dieser Umgebung bei einer greedy Policy beispielsweise nicht möglich gewesen.

Lisa Angold - 506007
Daniel Heising - 506603
Justus Pancke - 505533

Aufgabe 1b)

Ähnlich wie bei den State-Values konvergieren auch die Action-Values innerhalb weniger Zeitschritte deutlich , wie in Abbildung 1 zu sehen. Der wesentliche Unterschied zwischen Bellman mit State-Value Funktion und Bellman mit Action-Value Funktion ist der Einbezug der Wahl der Aktion.

Aufgabe 2

Siehe Code im Anhang.

Aufgabe 2b

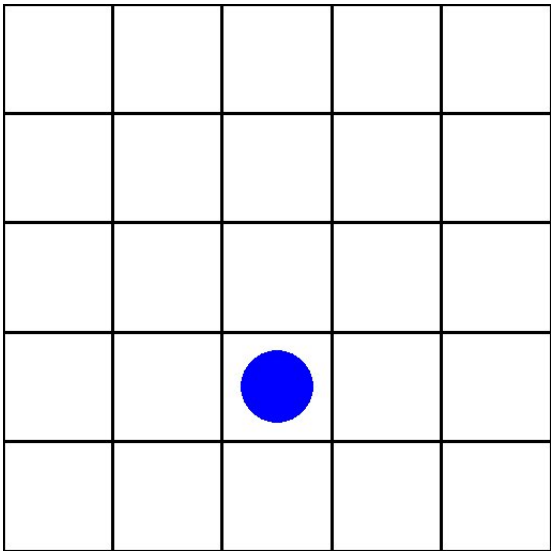

Abbildung 2: Schritt 0

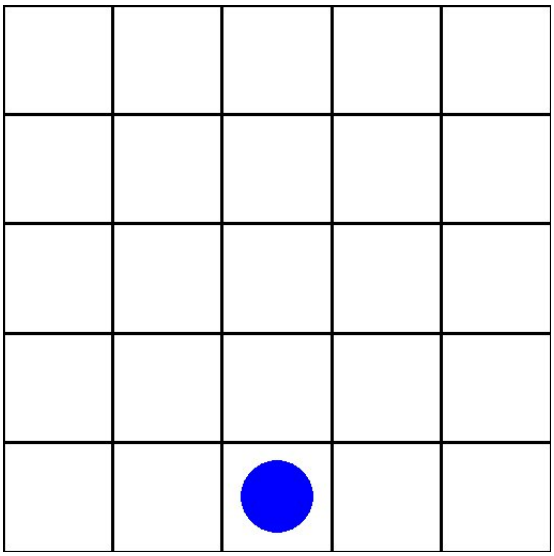

Abbildung 3: Schritt 1

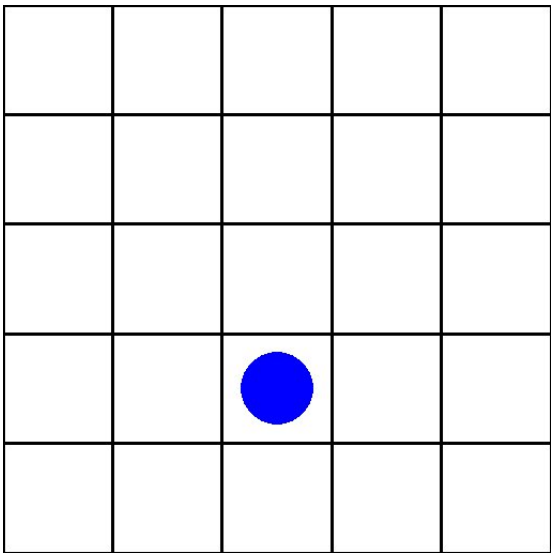

Abbildung 4: Schritt 2

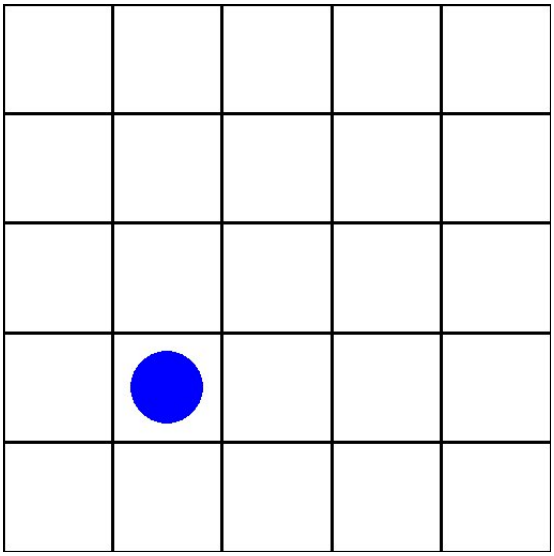

Abbildung 5: Schritt 3

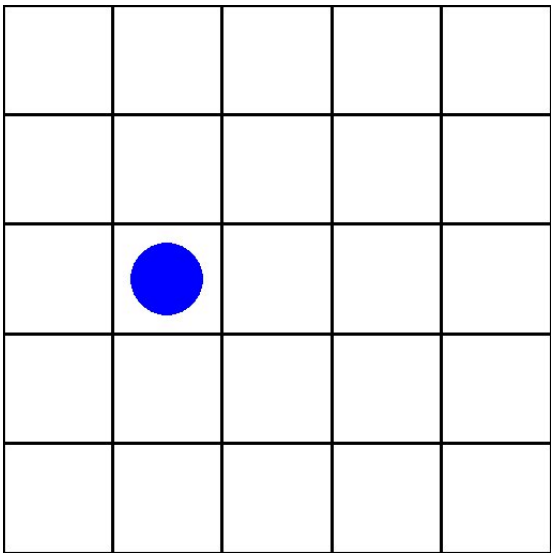

Abbildung 6: Schritt 4

Aufgabe 3

Aufgabe 3a)

Das MDP des CartPole Environments lässt sich formal als ein Tupel (S, A) definieren, mit

- S: Menge von Zuständen
- A: Menge von Aktionen
- P: Übergangsfunktion
- R: Belohnungsfunktion

Dabei wird die Zustandsmenge beschrieben durch vier verschiedene Eigenschaften $S = [CartPosition, CartVelocity, PoleAngle, PoleAngularVelocity]$ mit den Werten:

- $CartPosition \in [-4.8, 4.8]$
- $CartVelocity \in [-\infty, \infty]$
- $PoleAngle \in [-24°, 24°]$
- $PoleAngularVelocity \in [-\infty, \infty]$

Die Aktionsmenge wird beschrieben durch $A = 0, 1$ wobei $a = 0$ dafür steht, dass das Cart nach links geschoben wird und $a = 1$ dafür, dass es nach rechts geschoben wird.

Die Übergangsfunktion ist implizit gegeben durch die Simulation der CartPole Environment.

Das Cart wird zu jeder Episode auf einen Startzustand gesezt, indem alle Werte des Zustands einen Wert aus $[-0.05, 0.05]$ zugeordnet bekommen. Die Umgebungsdynamik ist dadurch definiert, dass solange Aktionen durchgeführt werden, bis eine der Terminierungsbedingungen gegeben ist:

- $PoleAngle < -12°$ oder $PoleAngle > 12°$
- $CartPosition < -2.4$ oder $CartPosition > 2.4$
- Es wurden 500 Zeitschritte durchlaufen

Die Belohnungsstruktur beschränkt sich auf eine Belohnung von $r = 1$ für jeden Zeitschritt inkl. des Schrittes der eine Episode terminiert. Das Ziel, das Pendel über mindestens 200 Zeitschritte aufrecht zu erhalten, setzt eine Gesamtbelohnung von mind. $G = 200$ für eine Episode voraus.

Aufgabe 3b)

Für den intuitiven Ansatz ohne Reinforcement Learning haben wir uns für fünf verschiedene Policies entschieden und diese miteinander verglichen.

1. `position` - Policy basierend auf $CartPosition$: Für $CartPosition < 0$ wähle $a = 0$, sonst $a = 1$. Dies bewirkt, dass das Cart nach rechts fährt, wenn es links von 0 steht und andersherum.
2. `angle` - Policy basierend auf $PoleAngle$: Für $PoleAngle < 0$ wähle $a = 0$, sonst $a = 1$. Dies bewirkt, dass das Cart nach links fährt, wenn die Pole nach links geneigt ist und nach rechts fährt, wenn das Pole nach rechts geneigt ist.
3. `angle_speed` - Policy basierend auf $PoleAngle$ und $CartVelocity$: Wenn $PoleAngle < 0$ und $CartVelocity > -1$ dann wähle $a = 0$, sonst $a = 1$. Somit bewegt sich das Cart nach rechts, wenn das Pendel sich nach rechts bewegt und das Cart gerade nach links fährt oder in einer sehr langsamen Geschwindigkeit nach rechts.
4. `angle_anglespeed` - Policy basierend auf $PoleAngle$ und $PoleAngularVelocity$: Wenn $PoleAngle + PoleAngularVelocity < 0$ dann wähle $a = 0$, sonst $a = 1$. Das Cart reagiert mit einer Bewegung nach links, wenn sowohl das Pendel nach links geneigt ist und nach links fällt als auch wenn das Pendel links geneigt steht und sich nur leicht nach rechts bewegt.
5. `angle_error` - Policy basierend auf letzter und vorletzter Beobachtung: Die Abweichung von $PoleAngle$ von dem Idealwert 0, welcher eine perfekte Balance widerspiegelt, wird berechnet. Außerdem wird die Abweichung von dem wert aus der vorherigen Iteration und dessen Abweichung wiederum zu dem wert der neuen Abweichung berechnet. Es wird überprüft, ob die zuletzt beobachtete Abweichung von $PoleAngle$ zusammen mit der stärker gewichteten beobachteten Abweichungsdifferenz größer ist als 0. Ist dies der Fall wird $a = 0$ gewählt, sonst $a = 1$. Diese Policy bewirkt, dass zuvor beobachtete Abweichungen stärker ins Gewicht fallen, sodass eher eine Aktion gewählt wird, die den Fehler ausgleicht.

Die Performance der verschiedenen Policies ist Abbildung 7 bis 11 zu entnehmen.

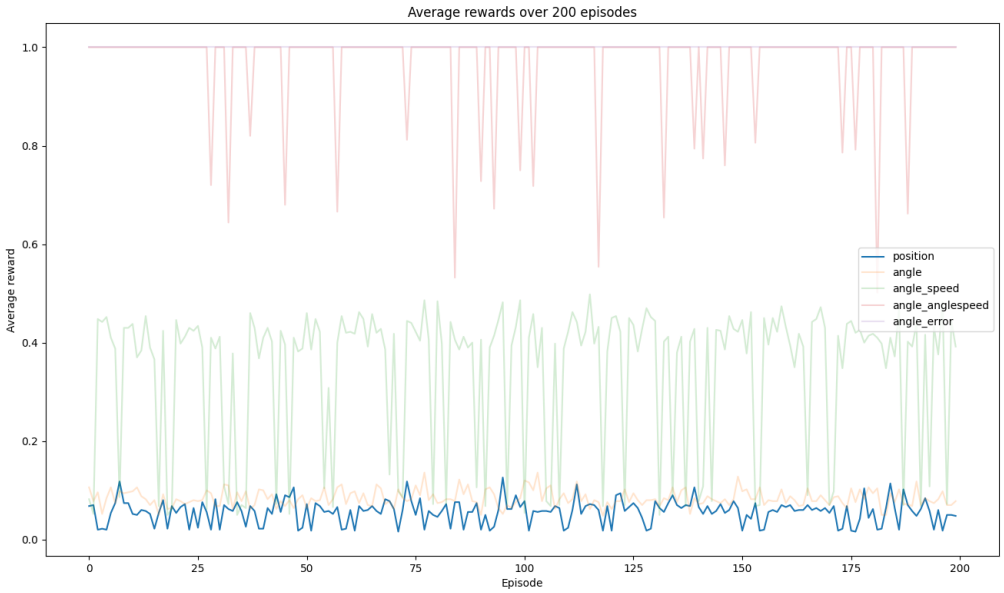

Abbildung 7: Durchschnittlicher Reward der Policy `position` über 200 Episoden.

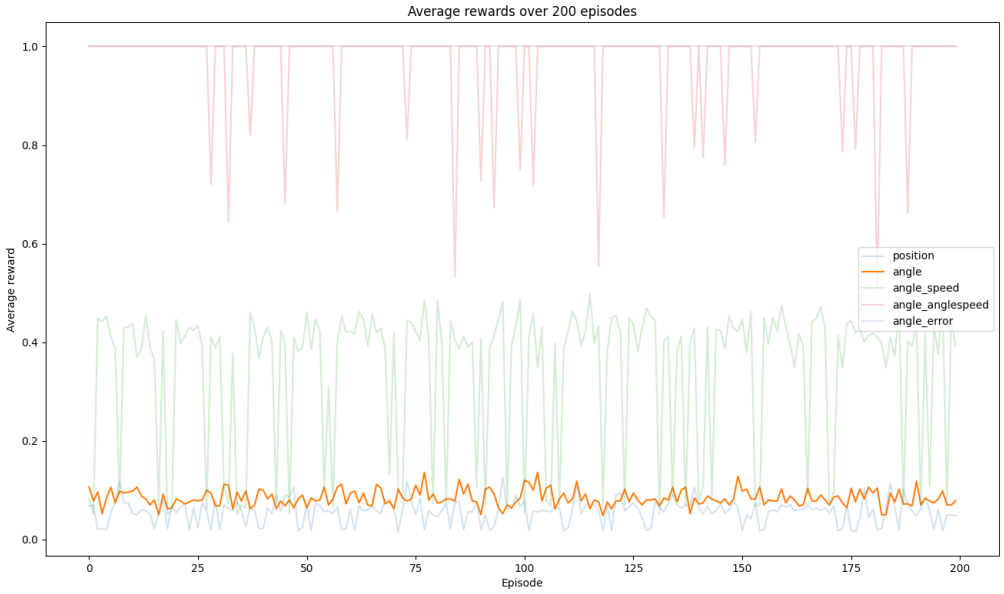

Abbildung 8: Durchschnittlicher Reward der Policy `angle` über 200 Episoden.

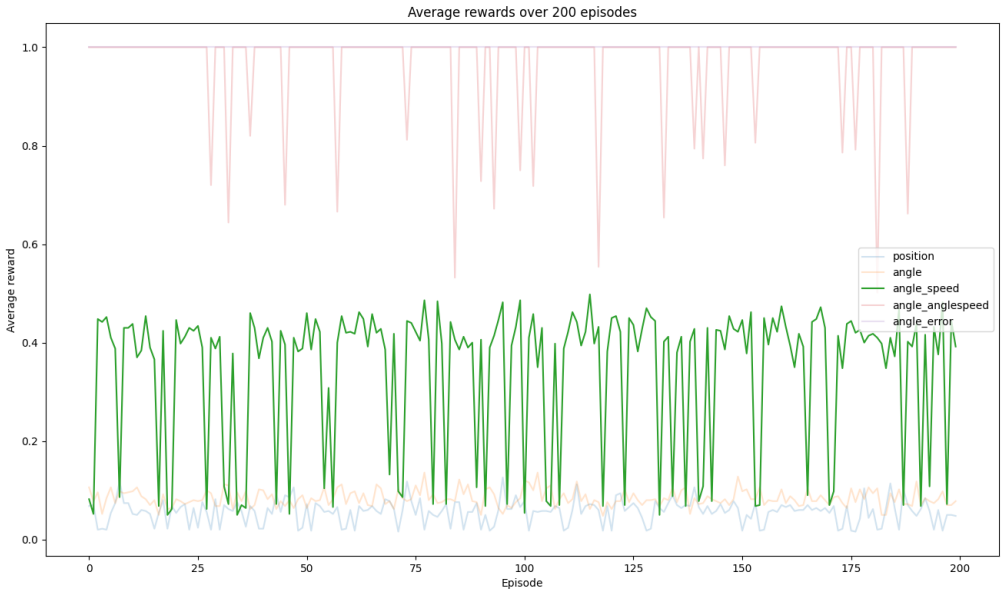

Abbildung 9: Durchschnittlicher Reward der Policy `angle_speed` über 200 Episoden.

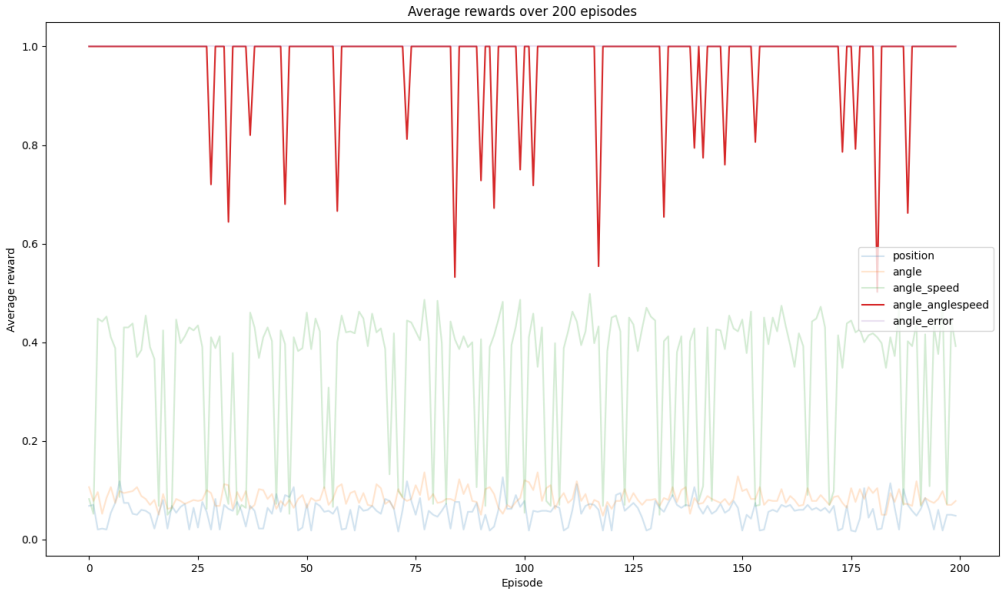

Abbildung 10: Durchschnittlicher Reward der Policy `angle_anglespeed` über 200 Episoden.

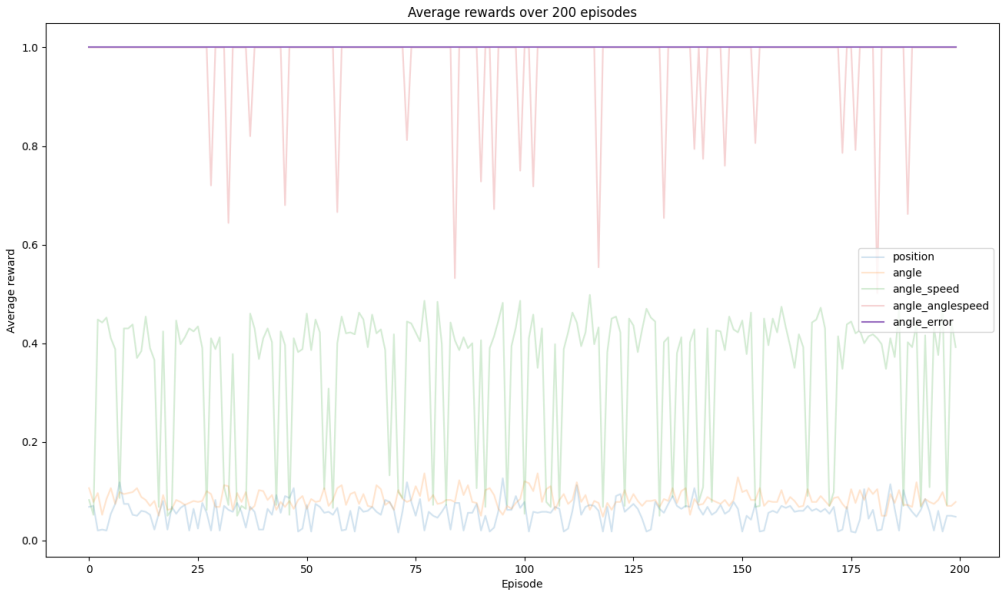

Abbildung 11: Durchschnittlicher Reward der Policy `angle_error` über 200 Episoden.

Anhand von Abbildung 7 und 8 lässt sich gut erkennen, dass die verschiedenen Variablen der Umgebung im Zusammenhang betrachtet werden müssen, da diese Policies oft früh beendet werden und sich so kein hoher kumulativer Reward bildet. Die Policies `angle_speed` (Abb. 9) und `angle_anglespeed` (Abb. 10) hingegen weisen einen deutlich höheren durchschnittlichen Reward über die Episoden vor. Vor allem `angle_anglespeed` weist eine gute Performance mit nur wenigen Ausreißern auf, was deutlich zeigt, dass eine Handlung basierend auf den Variablen *PoleAngle* und $PoleAngularVelocity$ vielversprechend ist. Das beste Ergebnis konnten wir mit der Policy `angle_error` (Abb. 11) erreichen, welche eine perfekte Performance verzeichnete.

Ein Probelm welches vor allem die Policies `position`, `angle` und `angle_speed` aufwiesen war die fehlende Effektivität bei längerer Betrachtung. Dadurch, dass immer nur der aktuelle Zustand betrachtet wurde, wurden Entscheidungen auch nur darauf basierend getroffen. Die Policy `angle_error` hat zusätzlich zum aktuellen Zustand auch die vorige Beobachtung miteinbezogen und diese stärker in die Entscheidung gewichtet. Dies könnte einer der ausschlaggebenden Faktoren sein, die eine durchgehend stabile Performance unterstützt haben.

Aufgabe 3c)

Wird der Reward genutzt, wie durch die Umgebung vorgesehen, trägt dieser Ansatz nichts zur Stabilisierung des Pendels bei. Durch einen stetigen Reward von +1 für jede Aktion, wird jede Aktion mit 1.0 bewertet. Dies führt dazu, dass die Aktionen für einen Agenten immer gleich wahrscheinlich sind und dadurch gerne die falsche ausgewählt wird. Dies würde sich lösen lassen, wenn der Reward tatsächlich beschreiben würde wie gut/schlecht die gewählte Aktion in Hinsicht auf den Gesamtverlauf ist. In Fig. 12 ist zu sehen, dass, falls ein Bandit überhaupt erreicht wird, der Reward immer linear ansteigt. Durch die Farbabstufungen ist ebenfalls zu erkennen, dass mit die-

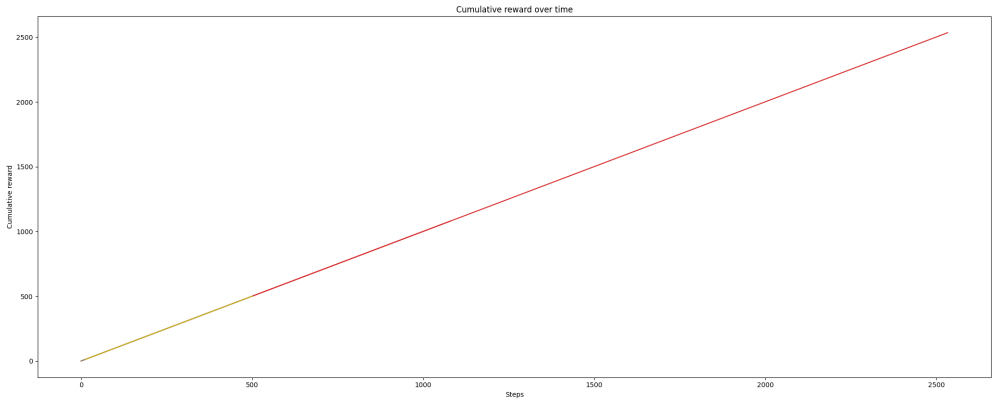

Abbildung 12: Kumulativer Reward der einzelnen Agenten über der Anzahl der Male, die sie ausgewählt wurden

sem Reward-Ansatz hauptsächlich ein Bandit erreicht wird und die anderen meistens nicht. Dies ist darauf zurückzuführen, dass die Episoden immer sehr schnell zu einer Abbruchbedingung geführt werden, da die Auswahl der Aktionen willkürlich ist.

Vermutlich führt eine höhere Diskretisierung (weniger Zustände) zu schnellerem Lernverhalten, aber zu wenig Stabilität, weil wenig Fälle abgedeckt sind. Andersherum wird das Lernverhalten langsamer, aber die Stabilität erhöht.