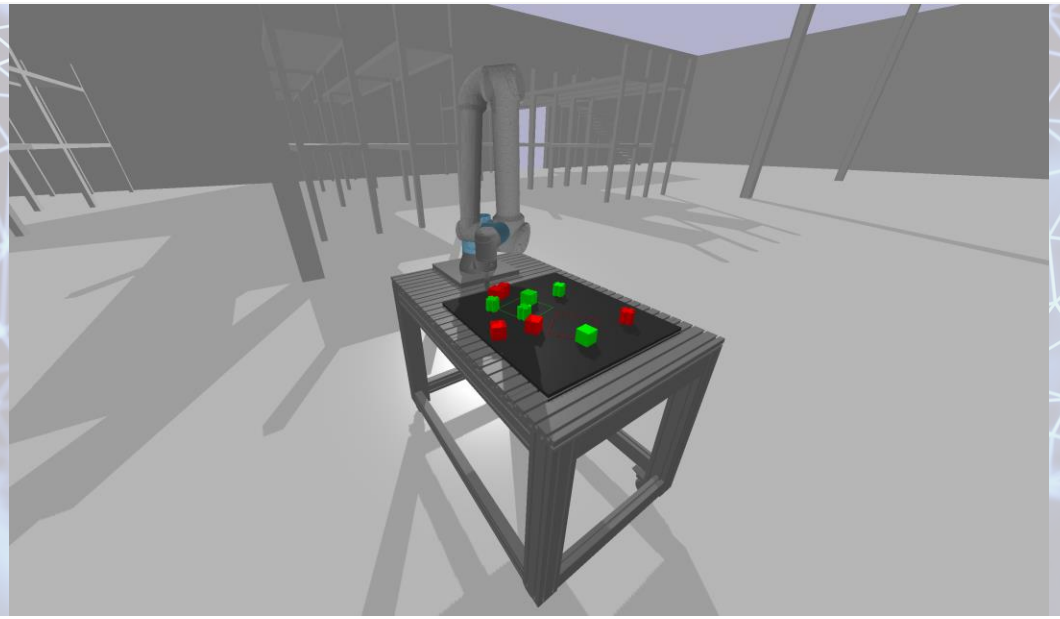
Hochschule Karlsruhe
University of
Applied Sciences

Fakultät für
Maschinenbau und
Mechatronik

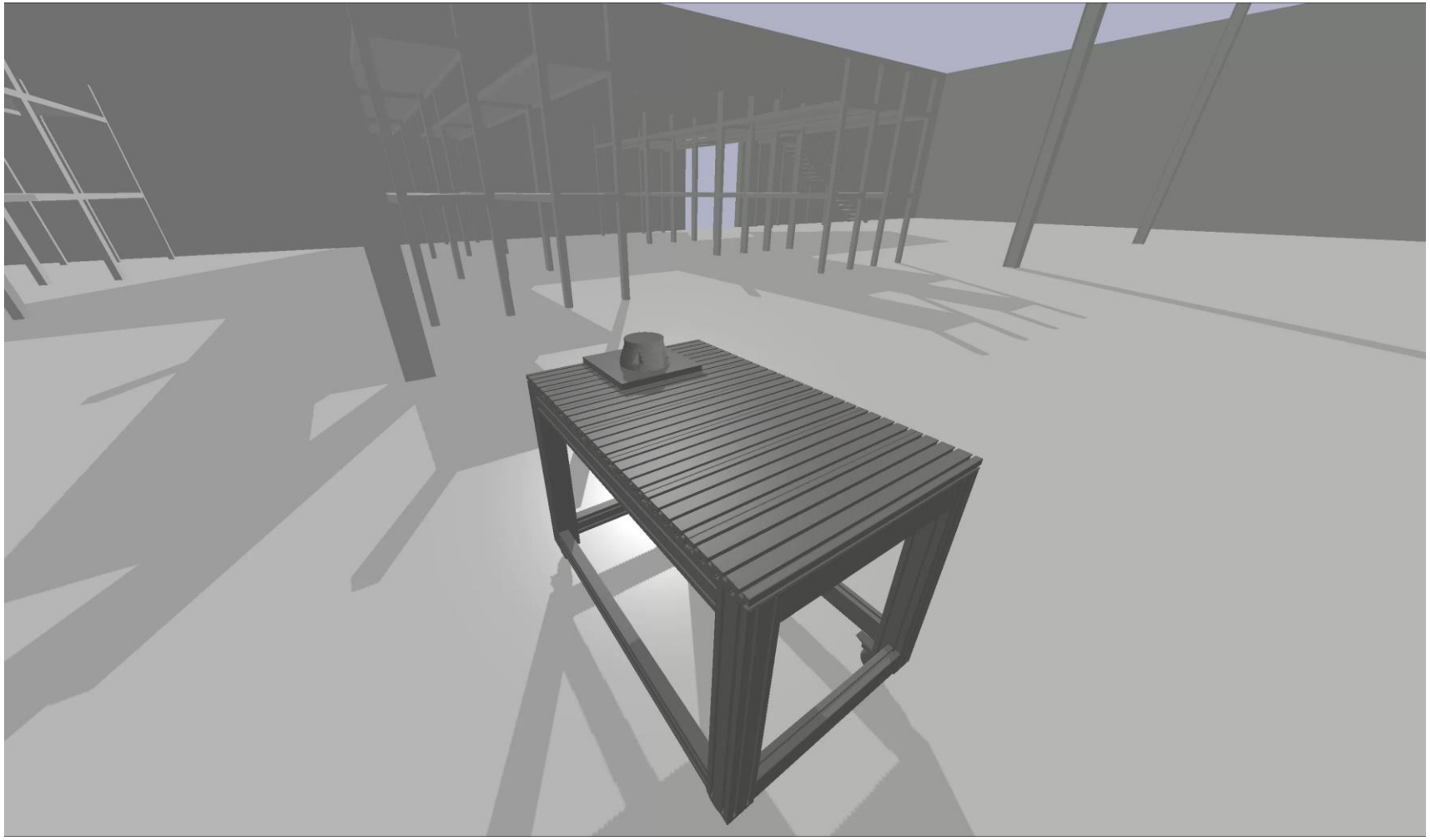
Roboterprogrammierung
Sorting via Pushing – Reinforcement Learning

Bearbeiter: Manuel Uibel
Philipp Herrmann
Tobias Unger

Betreuer: M. Sc. Gergely Sóti



Implementierung der Aufgabenstellung



Task:
“Sorting via Pushing: the workspace contains multiple objects of two different colors and areas corresponding to those colors. The task is for the robot to push all objects to their corresponding colored areas. The objects can be of different shapes, and the areas should be simple squares with colored borders able to fit all required objects. The objects and the areas should be placed randomly.“

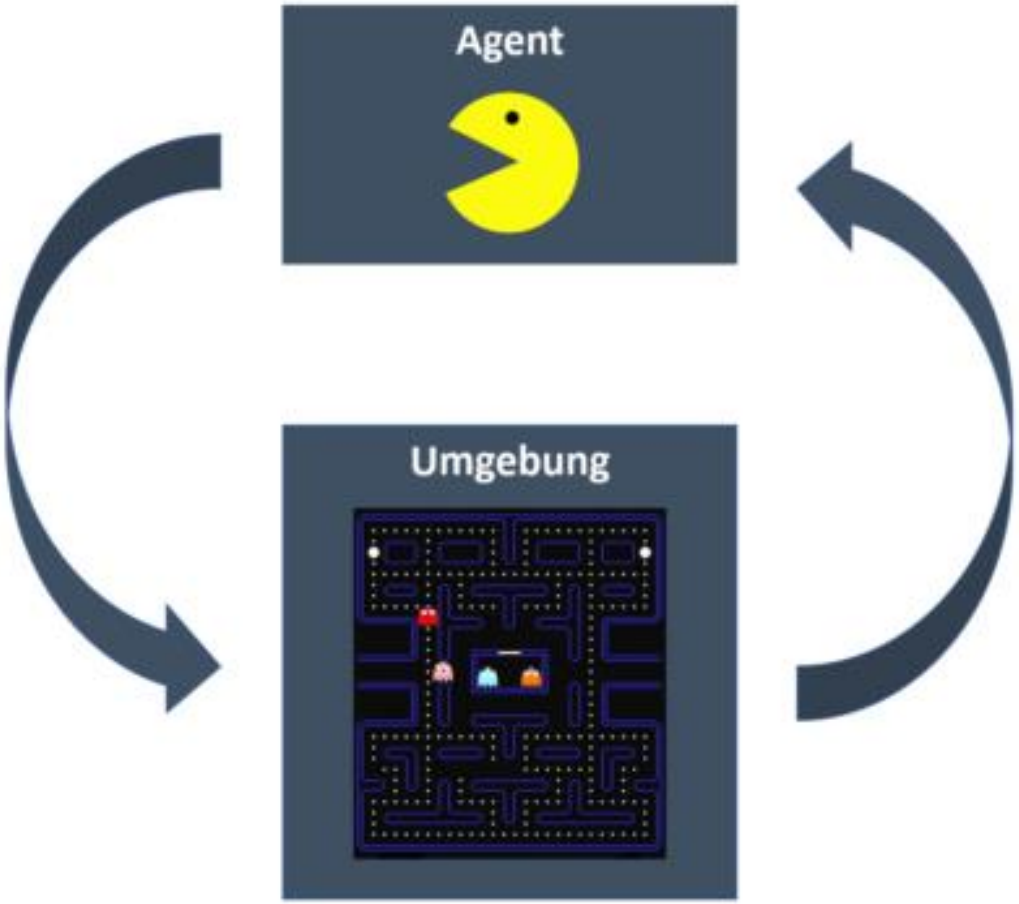
Warum Reinforcement Learning?
Vor- und Nachteile gegenüber Imitation Learning

+ Keine Demonstrationsdaten generieren
+ Kann gut mit neuen Situationen umgehen (→ Zufällige Platzierung der Objekte)
+ Kann gut mit Störgrößen umgehen
+ Überlegung am Anfang: Wir müssen nur gute Rewards definieren und das Model lernt den Rest von alleine. Ein Trugschluss?

- Hoher Rechenaufwand → genügend Zeit verfügbar
- Aufwendigere Implementierung → Datengenerierung aber auch aufwendig (vor allem Pushing kann sehr schlecht automatisiert erfolgen)

Reinforcement Learning

Action ∈ Action Space



Feedback:

- Observation Space
- Reward

Quelle: Wie funktioniert Reinforcement Learning? Bestärkendes Lernen erklärt - DataBraineo - Data Science Consulting

Action Space

- Action Space besteht aus 4 Actions (keep it simple):

- Move_left
- Move_right
- Move_forward
- Move_backward

- Jede Action verfährt den TCP in der jeweiligen Richtung um 10mm
- Schwierigkeiten: Drift aufgrund von Simulation (Diskretisierung, ect.) muss korrigiert werden

- Der Agent soll durch Rewards lernen, die für die derzeitige Observation passende Action zu berechnen.

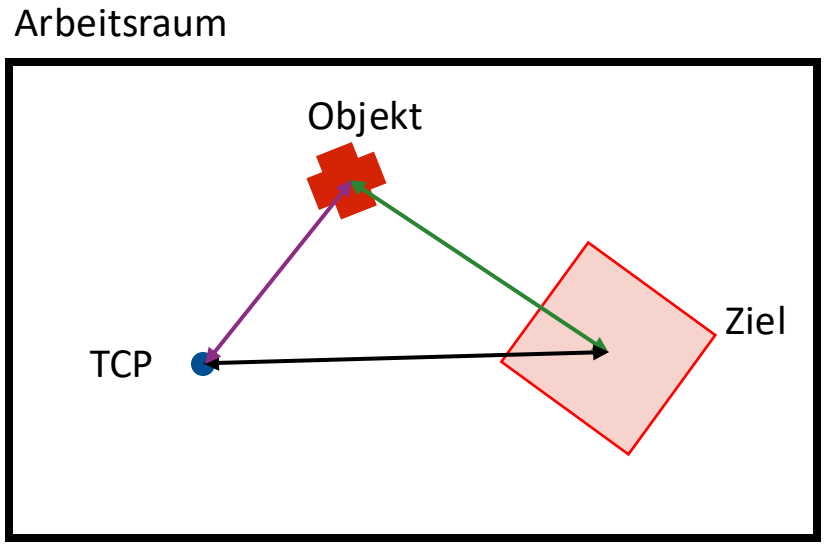
Observation Space

- Verschiedene Dinge ausprobiert:
- Observation Space besteht aus Positionen von TCP, Objects, Goals (flatted Array)
- Anpassungen:
  - Absolut- vs. Relativpositionen (→ TCP)
    → kein signifikanter Einfluss festgestellt
  - Alle Positionen vs. nur „relevante“ Positionen
    → schnellere Konvergenz
  - Normieren des Observation Space vs. keine Normierung
    → Signifikante Verbesserung der Konvergenz
  - Padding vs. kein Padding
    →Zuordnung zu Farben
- Nur durch einen sinnvollen Observation Space ist gesichert, dass der Roboter sich zuverlässig auf das Objekt zu bewegt (das Schieben ist komplizierter).

Rewards

Grundlegende Idee:

- Aktion des Roboters bewerten
  - Belohnt wird:
    - TCP nähert sich Objekt ↔
    - Objekt wird in Richtung Ziel geschoben ↔
    - Objekt ins Ziel geschoben □
  - Bestraft wird:
    - TCP entfernt sich von Objekt ↔
    - Objekt entfernt sich vom Ziel ↔



Nearest Object

2 Optionen:

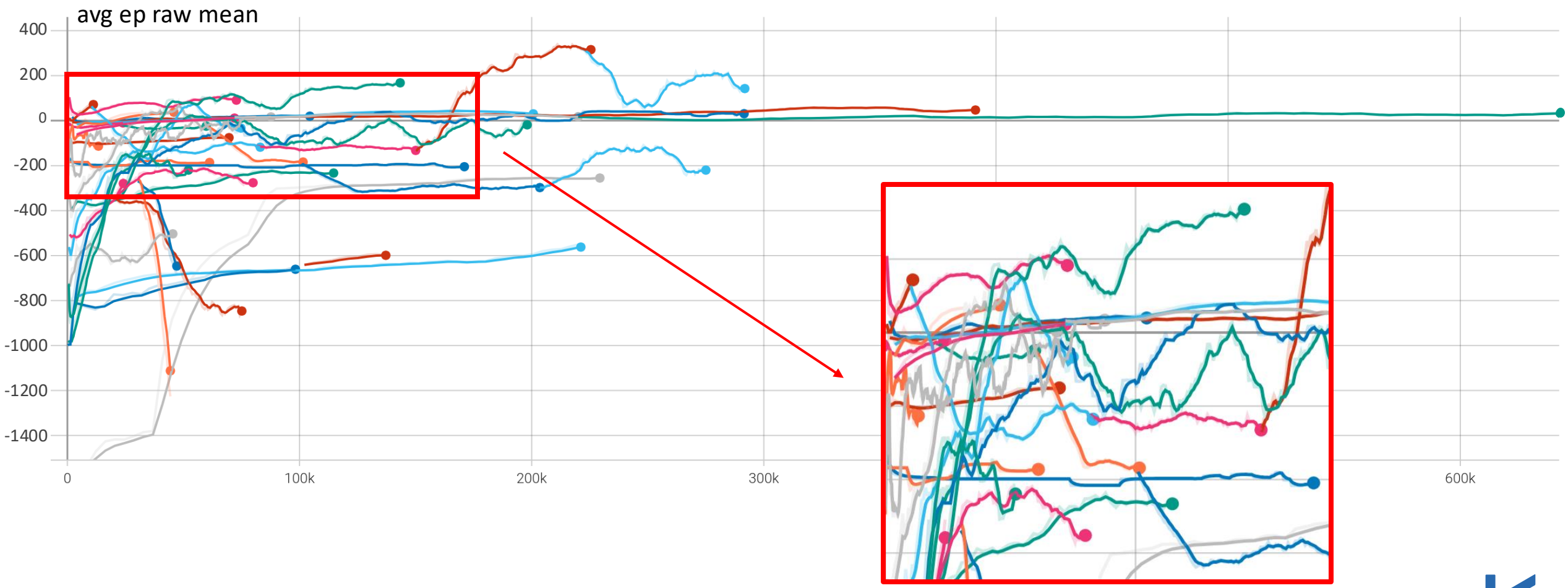
- Erlaube Objektwechsel:
  - Prüfe ob Objekt im Ziel
  - Berechne minimale Distanz von allen Objekten zum TCP

- Ausgewähltes Objekt muss ins Ziel:
  - Unterscheide zwischen Anfangs- / Endschritt und Zwischenschritt

↓

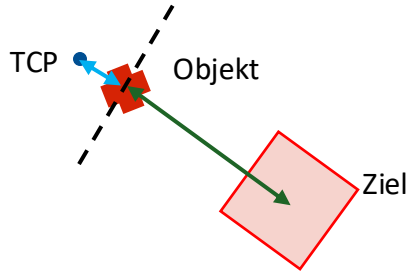
- Prüfe ob Objekt im Ziel
- Berechne Distanz von aktuellem Objekt zum TCP

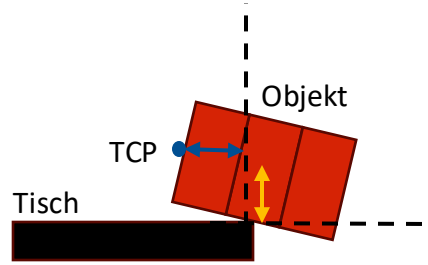
Logging mittels Tensorboard



Ablauf eines Schrittes

- Vorhergesagte Aktion ausführen
- Prüfe Endbedingungen
  - offTable
  - offWorkarea
  - Timeout
  - finished
- Berechne Reward
- Observiere neue Umgebung
- Rückgabe von:
  - Reward
  - Umgebung
  - Endzustände



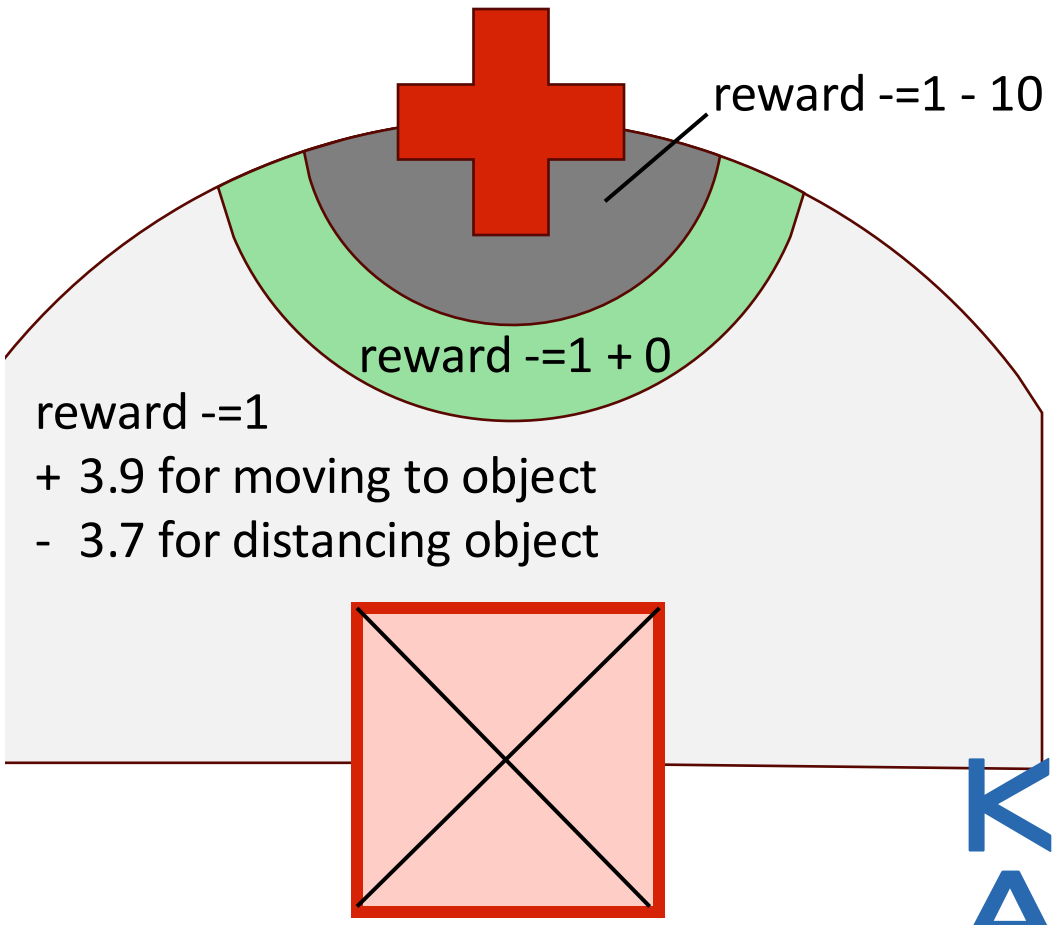


TCP-X	TCP-Y	OB1-X	OB1-Y	OB1-φ	...	GO1-X	GO1-Y	GO1-φ	...

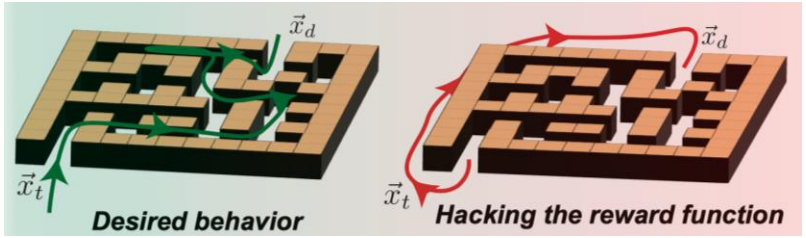
Beobachtungen zu Rewards und Nebeneffekten
Reward-Engineering

time penalty: -= 1

- TCP will „durch“ Objekt laufen
- TCP nähert sich Objekt von Seite des Ziels an
- Zick-Zack-Bewegungen des Roboters hinter Objekt
- Problem:
  - Reward Hacking → Roboter lernt ungewollte Strategie zur Maximierung des Rewards



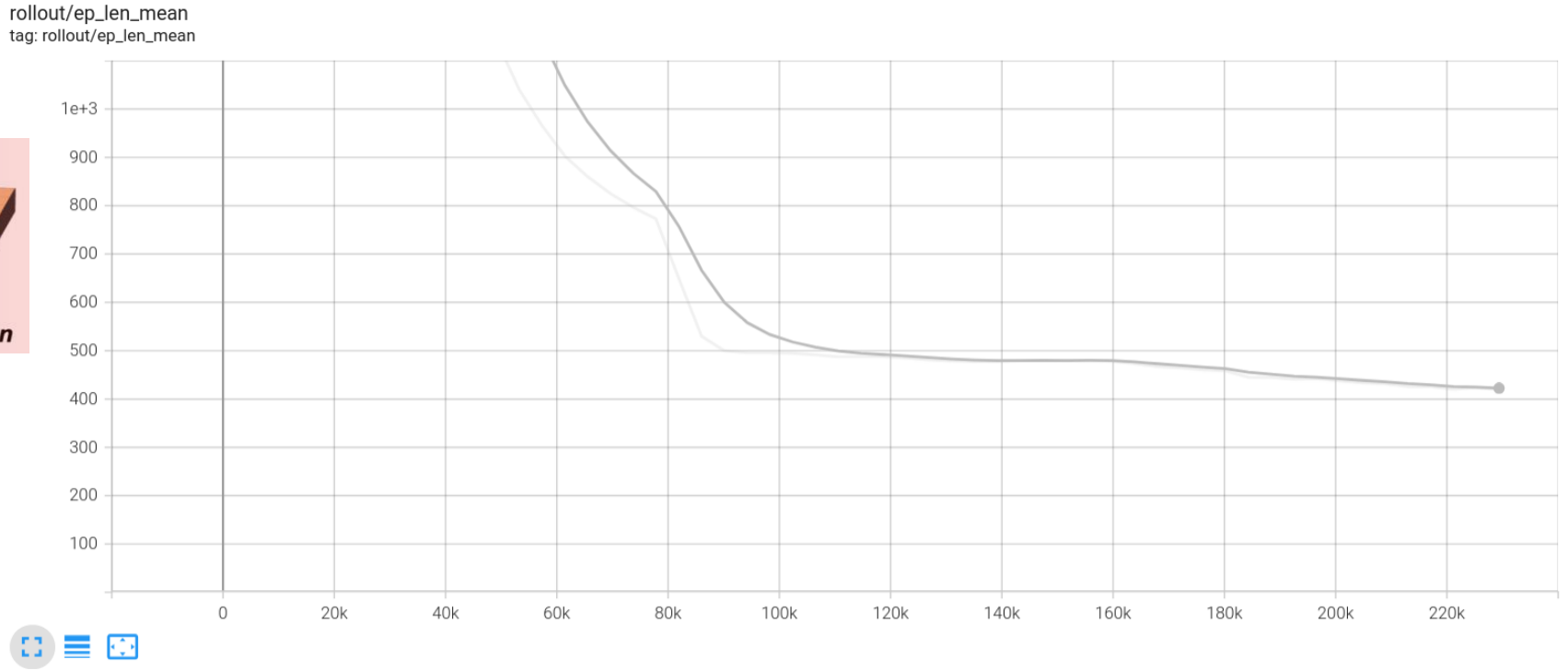
Reward Hacking

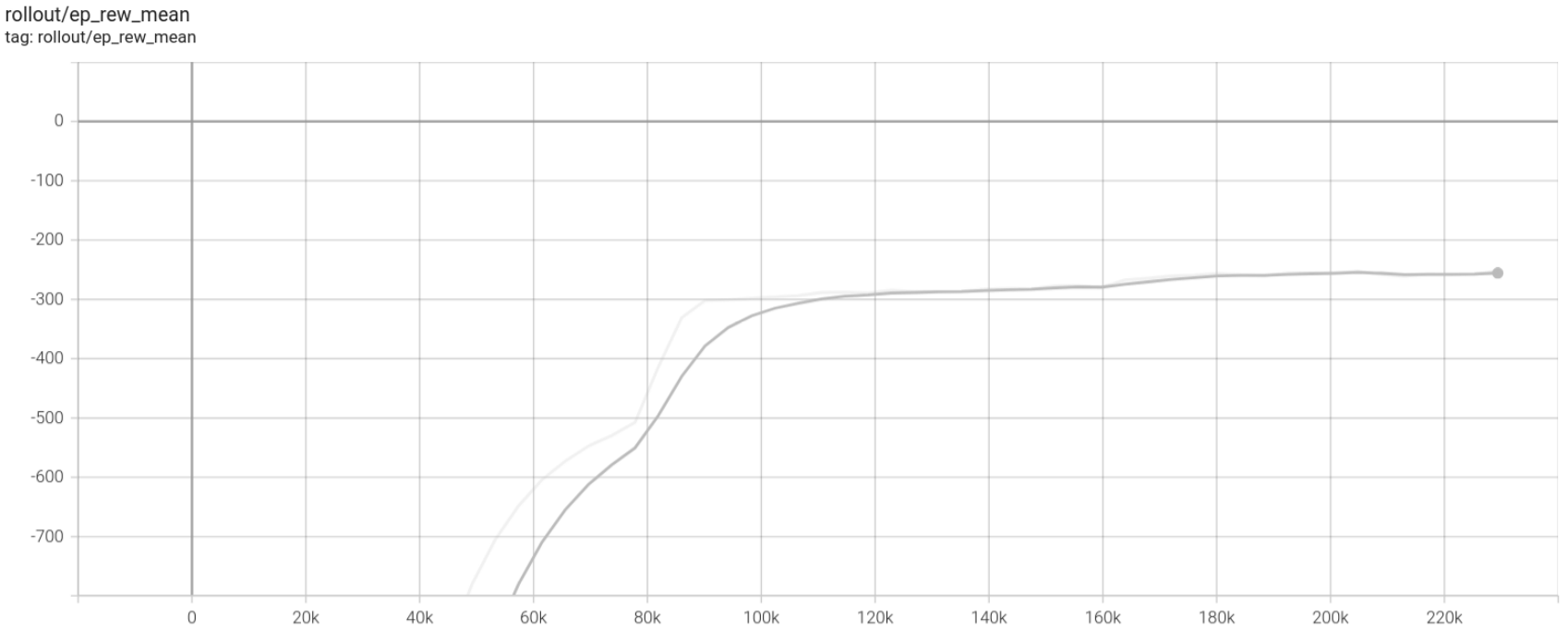


Quelle: Designing Societally Beneficial Reinforcement Learning Systems

Beispiel für Reward Hacking:

- Reduzierung der Episodenlänge durch ObjectOffTable Methode
- Durch Rewardanpassungen kann dies verhindert werden
- z.B. ObjectOffTable
  → Reward = - 100





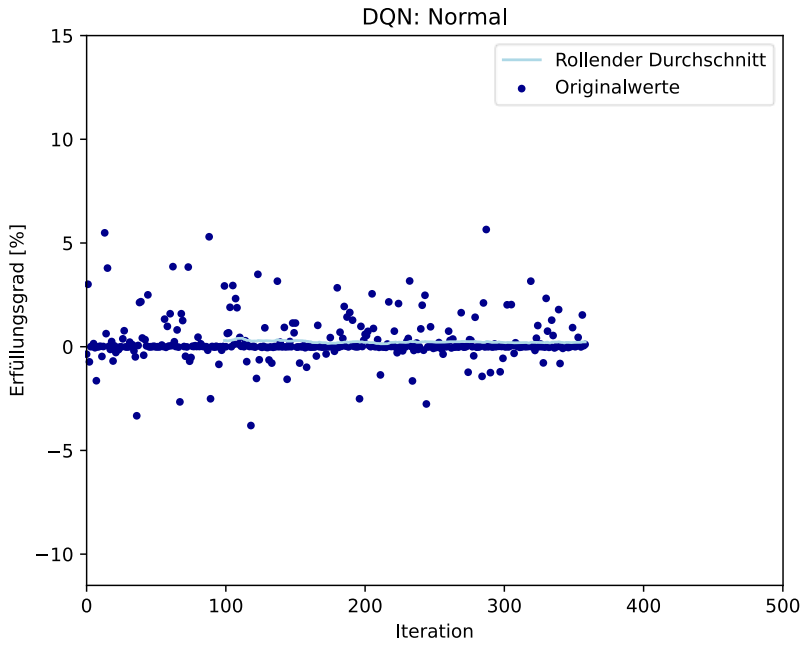
Modelle / verwendete RL-Algorithmen

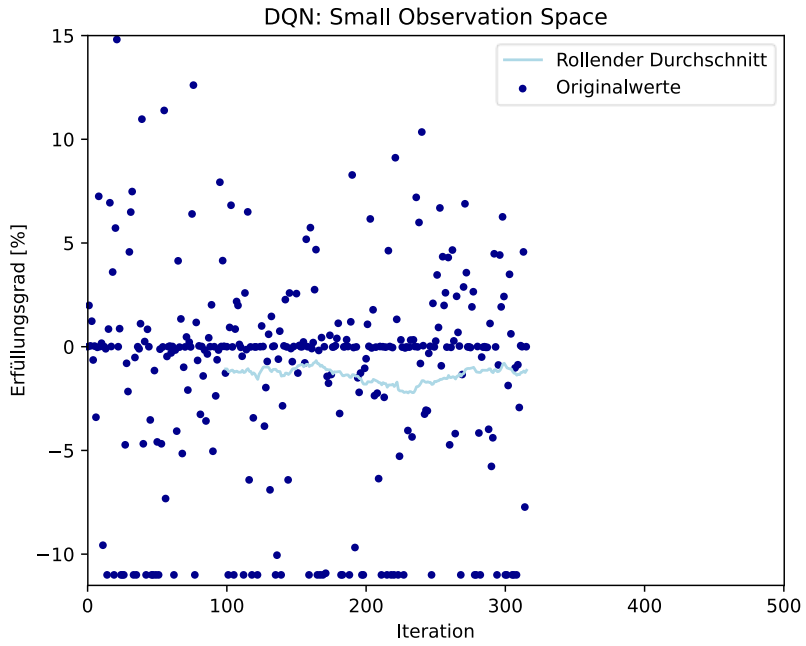
- PPO (Proximal Policy Optimization):
  - Für erste Versuche erfolgreich verwendet
  - robustes, stabiles Modell; erfordert keine Anpassungen
  - auch kontinuierliche Action Spaces möglich
  - höherer Rechenaufwand als DQN
- DQN (Deep Q-Network):
  - Vergleichsalgorithmus; später kompletter Wechsel
  - optimiert für Verwendung der GPU → **deutlich schnellere Berechnung**
  - auf diskrete Aktionsräume beschränkt / optimiert (vor / zurück / links / rechts)
- Custom MLP Policy eingerichtet, aber wieder verworfen
  - 2 Layer, 256 Knoten
  - Standard MLP Policy müsste ausreichend sein (2x64)
- Hyperparameter für PPO und DQN angepasst, z.B. Exploration Rate, …

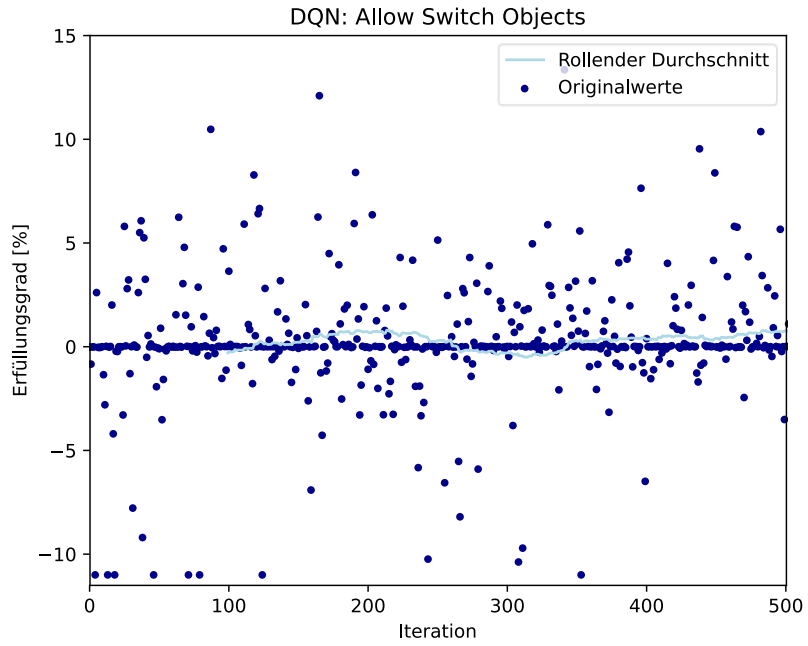
Ausführbare Skripte

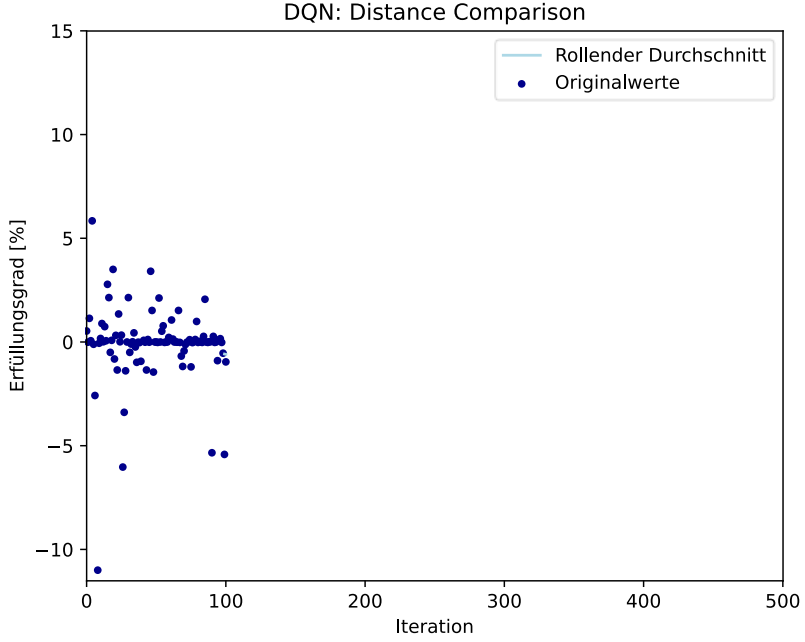
- Projektstruktur / Funktionen

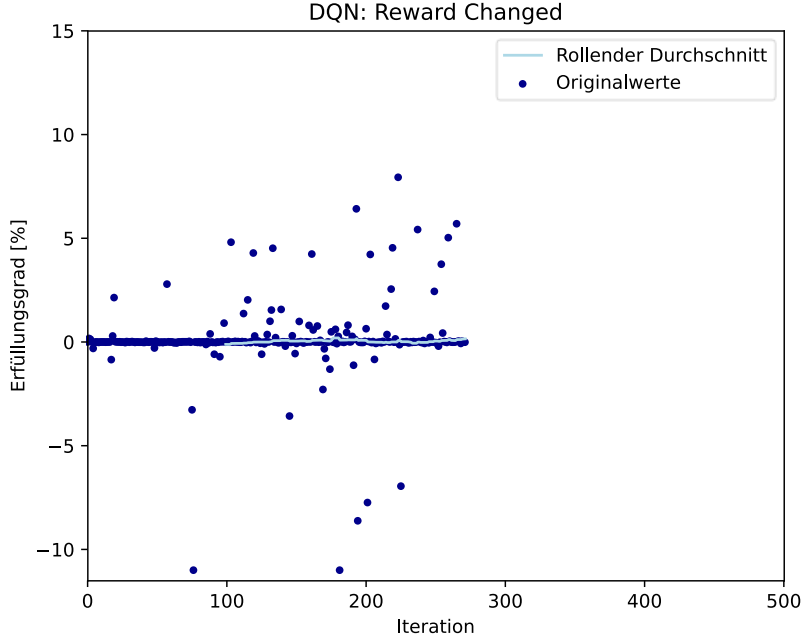
Skriptname	Funktion/Aufgabe
checkEnv.py	Roboter lässt sich mit wasd manuell steuern Erprobung des Environments, Testen von z.B. Rewardfunktionen
sortingLearn.py	Trainieren der Modelle; Auswahl des Modelltyps Erstellen und Speichern von Logs und Model.zip´s
pushToSort.py	Testen der trainierten Modelle, Predict für Actions
plotScores.py	Plotten der Scores

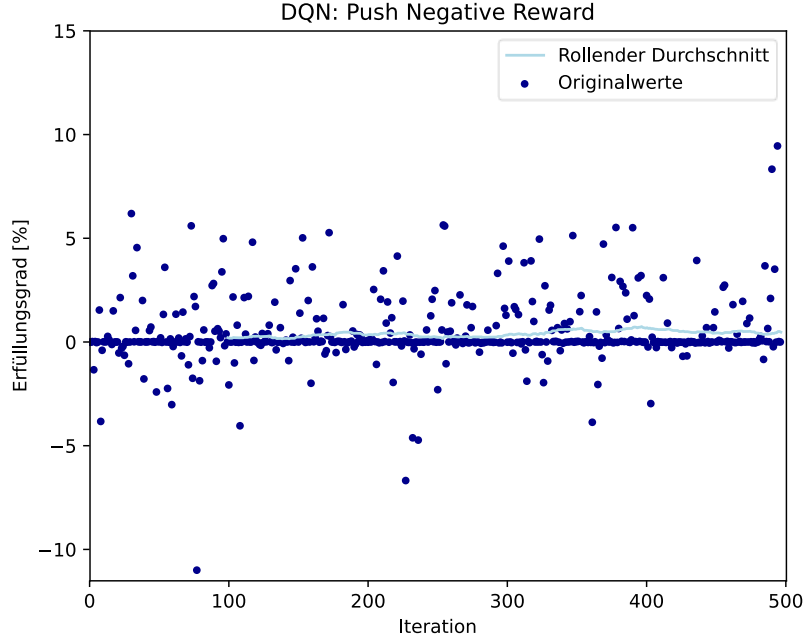

DQN: Normal
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration


DQN: Small Observation Space
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration


DQN: Allow Switch Objects
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration


DQN: Distance Comparison
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration


DQN: Reward Changed
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration


DQN: Push Negative Reward
Rollender Durchschnitt
Originalwerte
Erfüllungsgrad [%]
Iteration

Fazit

- Implementierung des Environments erfolgreich abgeschlossen
  - Alle benötigten Funktionen zum Spawnen der Objekte, Generieren des Observation Space, etc. funktionieren.
  - Keine größeren Schwierigkeiten bei der Implementierung

- Trainieren, Laden, Auswerten, Testen der Modelle
  - Umsetztung mittels Tensorboard, Score-Logging hat gut funktioniert
  - Funktioniert gut, die Auswertung zeigt jedoch keinen hohen Erfüllungsgrad (zurecht).

- Sortieren der Objekte
  - Objekte werden nicht sortiert, Agent hat nicht gelernt die Objekte zuverlässig ins Ziel zu schieben. Annähern des TCP ans Objekt funktioniert.

  → evtl. Optimierung der Rewards, längere Trainingsdauer

Ausblick

- Repo wird zur Verfügung gestellt (lauffähig in enthaltenem Docker-Container)
- README.md zur Bedienung und Erklärung wird zur Verfügung gestellt
- Evtl. Folgeprojekte möglich? Mögliche Verbesserungen:
  - Optimierung der Rewards
  - Signifikante Erhöhung der Trainingsdauer

Quelle: Ausblick 2022 - Fairtrade Town Geldern