

Technische Hochschule Ulm
Fakultät Produktionstechnik und Produktionswirtschaft



Masterprojektarbeit zum Thema:

Produktionssteuerung mit selbstlernenden Multiagentensystemen

Umsetzung anhand eines virtuellen Simulationsmodells

Autor: Elias J. Zipfel
Betreuung: Prof. Dr. Sven Völker
E-Mail: elias.zipfel@gmx.de
Matrikelnummer: 3136577
Studiengang: Systems Engineering und Mngt.
Abgabe: 21.03.2022

Inhalt

Inhalt.....	1
1. Einleitung	2
2. Anwendung von Begriffen und Methoden.....	4
2.1. Produktionsplanung und -steuerung (PPS)	4
2.2. Multiagentensystem.....	5
2.3. Selbstlernend - künstliche Intelligenz (KI).....	7
2.4. Simulation	9
3. Konzeption und Umsetzung des Simulationsmodells	10
3.1. Beispiele aus der Literatur	10
3.2. Gestaltung des fiktiven, physischen Produktionssystems	12
3.3. Gestaltung des Multiagentensystems	14
3.4. Heuristiken zur Produktionssteuerung	16
3.5. Anpassung bestehender Softwarekomponenten.....	19
3.6. Simulationssteuerung und Berechnungszeit.....	21
4. Experimentieren mit dem Simulationsmodell	23
4.1. Erste Phase: Nachweis der Lernfähigkeit.....	24
4.2. Zweite Phase: Komplexitätssteigerung	26
4.3. Dritte Phase: Möglichkeiten zur Leistungssteigerung	28
4.4. Kritik am Simulationsmodell	30
5. Zusammenfassung und Ausblick	31
Literaturverzeichnis	32

1. Einleitung

Variablere Nachfragemengen und heterogenere, individualisierte Kundenbedürfnisse – in diese Richtung entwickelten sich in den letzten Jahrzehnten die Kundenbedürfnisse (Abb. 1). Da erfolgreiche Geschäftsmodelle maßgeblich von den sie umgebenden Kundenbedürfnissen und Marktkräften abhängen, macht eine Veränderung dieser Faktoren eine Anpassung der Geschäftsmodelle notwendig.

Diese Entwicklung führt zu einem starken Anstieg an Komplexität im Umfeld von Unternehmen. Der wachsenden äußeren Komplexität müssen ausreichend komplexe unternehmensinterne Strukturen entgegengestellt werden, um flexibel auf die dynamischen Marktbedürfnisse reagieren zu können. Dies hat eine Struktur mit einem höheren Grad an Dezentralisierung und Autonomie zur Folge. Gleichzeitig muss jedoch weiterhin die Wirtschaftlichkeit durch das Nutzen von Skaleneffekten und Leistungsoptimierung sichergestellt werden. (Bauernhansl 2014, S. 36)

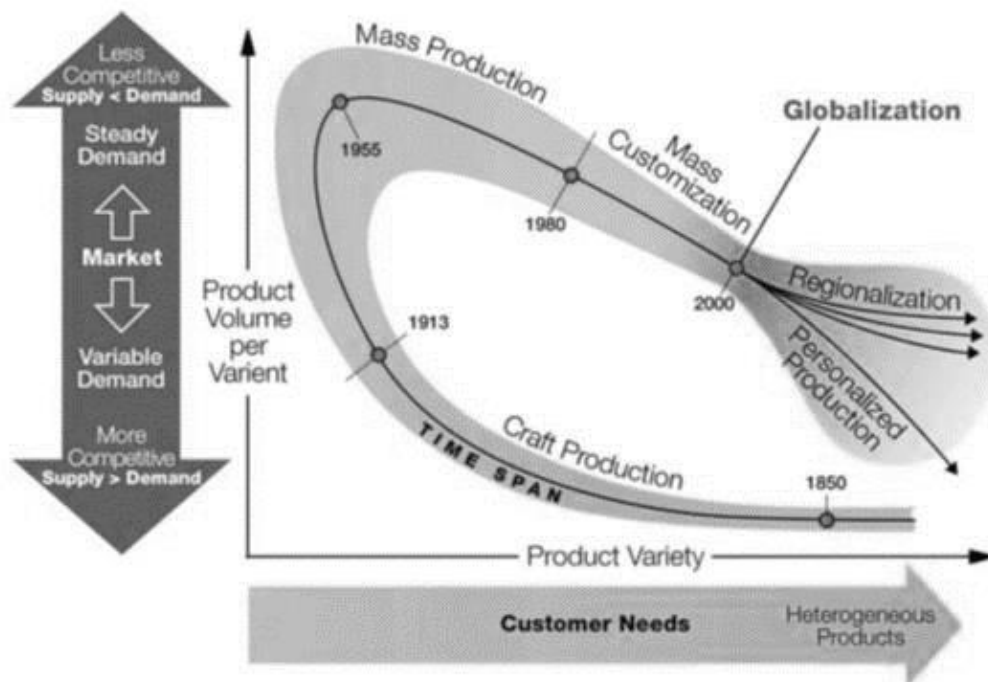


Abbildung 1: The Global Manufacturing Revolution (Koren 2010, S. 34)

Für Unternehmen, die physische Produkte produzieren, ist hierbei insbesondere die Produktionsplanung und -steuerung gefordert, dem Bedarf nach Flexibilität gerecht zu werden. Dezentralität und Autonomie können hier beispielsweise durch eine agentifizierte Struktur, in der kleine Einheiten selbst entscheiden, abgebildet werden (Scholz-Reiter und Höhns 2006, S. 746). Das Systemverhalten fügt sich dabei aus den Einzelentscheidungen der Agenten zusammen.

Die Intelligenz der Agenten in einem solchen Multiagentensystem kann auf verschiedene Arten implementiert werden. Eine Möglichkeit hierfür ist die Verwendung maschinellen Lernens, genauer von *Bestärkendem Lernen* mit einem *Künstlichen Neuronalen Netz* zur Abbildung des erzeugten Wissens.

Ziel der Arbeit ist es, Erkenntnisse zu gewinnen, ob ein solches selbstlernendes Multiagentensystem eine leistungsfähige Lösung für den Einsatz in der Produktionssteuerung sein kann. Hierzu wird ein derartiges System virtuell simuliert und dessen Leistungsfähigkeit geprüft. Bei erfolgreicher Umsetzung eines ausreichend realitätsnahen Simulationsmodells lässt sich annehmen, dass sich aus den Ergebnissen auch Schlussfolgerungen für reale Verhältnisse ableiten lassen. Die Implementierung des selbstlernenden Algorithmus baut dabei auf einen an der Technischen Hochschule Ulm entwickelten Softwareagenten für die Produktionssteuerung auf.

In Kapitel 1 werden die für die Arbeit notwendigen Begriffe und Methoden erläutert. Es wird ausgeführt, wie diese verwendet werden und worauf der Fokus gerichtet ist.

Kapitel 2 beschäftigt sich mit der Konzeption und Umsetzung des virtuellen Simulationsmodells. Es werden Beispiele aus der Literatur als Ideengeber untersucht. Die Bestandteile des Modells werden entsprechend der bestehenden Anforderungen entwickelt. Dabei werden bestehende Softwarekomponenten soweit möglich genutzt und sofern nötig ergänzt.

In Kapitel 3 werden die durchgeführten Experimente und deren Ergebnisse ausgeführt und Schlussfolgerungen hinsichtlich der Leistung gezogen. Dabei wird das Simulationsmodell den Bedürfnissen entsprechend iterativ weiterentwickelt.

2. Anwendung von Begriffen und Methoden

Im Folgenden werden die der Arbeit zugrundeliegenden Begriffe und Methoden definiert. Darüber hinaus erfolgt deren Anwendung auf die Bestandteile des Simulationsmodells. Insbesondere wird dabei auf die Titelbestandteile „Produktionssteuerung“, „selbstlernend“ und „Multiagentensystem“ eingegangen.

2.1. Produktionsplanung und -steuerung (PPS)

Die Produktionsplanung und -steuerung hat als Aufgabe, einen reibungslosen und wirtschaftlichen Produktionsprozess sicherzustellen. Unter dem Begriff werden in der Literatur verschiedene Konzepte beschrieben. Diese Konzepte unterscheiden sich durch die jeweilige Unterteilung der notwendigen Schritte, um die Gesamtaufgabe zu erfüllen. Bisher konnte nach Meudt et al. Kein vorherrschender Konsens bezüglich der Einteilung und deren Benennung gebildet werden. Für einen groben Überblick lassen sich Konzepte, die die Integration anderer Informationssysteme betonen, von solchen mit einer ausgeprägt hierarchischen Struktur und einem größeren Fokus auf die Produktionssteuerung unterscheiden. (Meudt et al. 2017, 1-10)

In dieser Arbeit bietet die Anwendung eines Konzepts mit größerem Fokus auf die Produktionssteuerung den Vorteil, dass die durch das selbstlernende Multiagentensystem zu erfüllende Teilaufgabe damit besser beschrieben werden kann. Angewandt wird deshalb die von Meudt et al. nach dem Vergleich mehrerer Definitionen vorgeschlagene Unterteilung (Meudt et al. 2017, S. 7). Diese ist unter Verwendung von Synonymen auch deckungsgleich mit der Unterteilung nach dem Gabler Wirtschaftslexikon (Siepermann 2018).

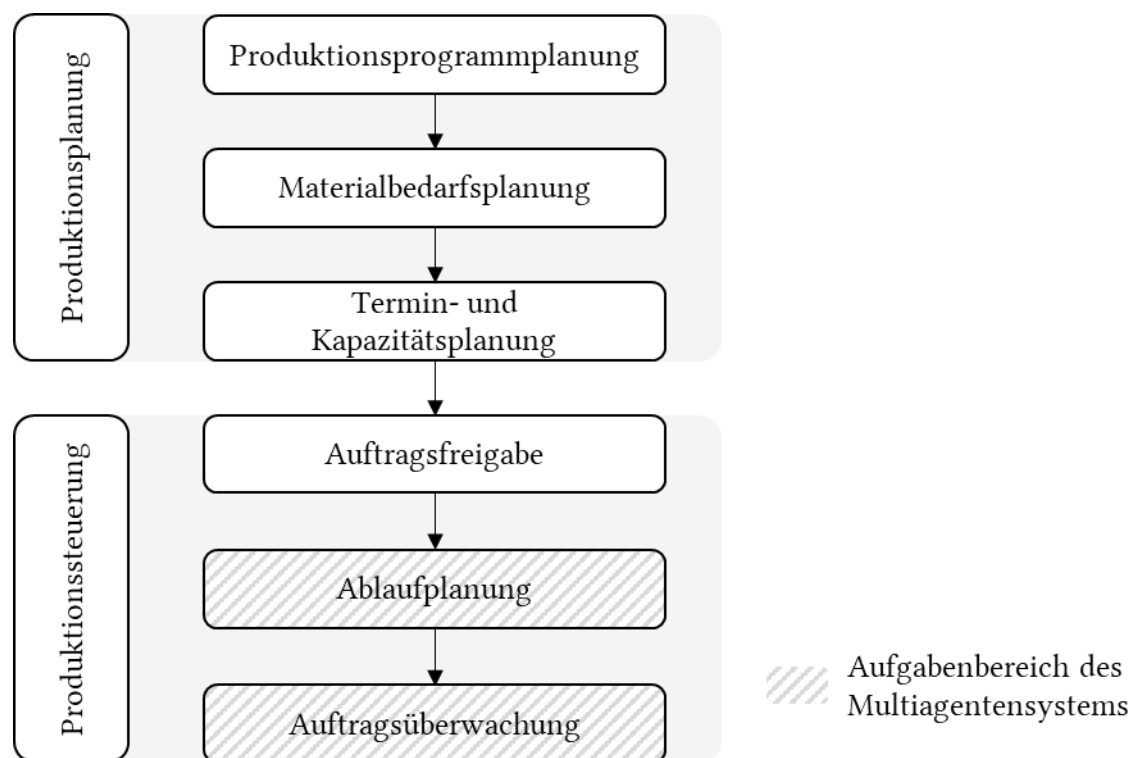


Abbildung 2: Teilaufgaben der PPS

Der Übergang von der Planung in die Produktionssteuerung wird hierbei durch die Auftragsfreigabe gekennzeichnet. Die Aufträge werden nach Prüfung der Verfügbarkeit von Material und Kapazitäten zur Fertigung freigegeben. Im Rahmen dieser Arbeit übernimmt das selbstlernende Multiagentensystem die anschließende Ablaufplanung und Auftragsüberwachung (vgl. Abbildung 2). Detaillierter betrachtet bedeutet dies, dass das System die genauen Bearbeitungstermine auf den einzelnen Bearbeitungsstationen des Produktionssystems festlegt und den Fertigungsablauf bei Störungen korrigiert (Siepermann 2018). Ausgangspunkt für das Simulationsmodell sind damit die zur Fertigung freigegebenen Aufträge.

Ziel der Arbeit ist es, herauszufinden, ob der Einsatz des selbstlernenden Multiagentensystems eine leistungsfähige Lösung ist. Um die Leistung zu bewerten, können verschiedene Leistungskriterien angewandt werden. Denkbare Größen sind hierbei beispielsweise die Zykluszeit, der Durchsatz, die Kapazitätsauslastung, die Umrüsthäufigkeit, die Termintreue oder die Durchlaufzeit. Untereinander stehen diese Größen in Beziehung und teilweise entstehen Zielkonflikte. Aus diesem Grund ist es in der Praxis für ein vollständiges Bild meist notwendig, eine Bewertung anhand mehrerer Kriterien gleichzeitig vorzunehmen. In dieser Arbeit wird zur Vereinfachung jedoch der Durchsatz als einziges Leistungskriterium herangezogen.

Zur Beurteilung der Leistungsfähigkeit des selbstlernenden Multiagentensystems ist es notwendig, neben dem Leistungskriterium Durchsatz auch Vergleichssysteme zu definieren. In Relation zu diesen kann die Leistungsfähigkeit beurteilt werden. In diesen Vergleichssystemen wird die Intelligenz der Agenten durch klassische, d.h. nicht selbstlernende Heuristiken abgebildet. Die Bestimmung absoluter Leistungsoptima ist aufgrund der vielen dynamischen Einflussfaktoren bereits bei überschaubaren Produktionsprozessen nicht praktikabel möglich (Claus et al. 2021, S. 4). Somit kann die Leistung nicht mit dem Optimum abgeglichen werden. Es muss in jedem Fall auf Heuristiken zurückgegriffen werden.

2.2. Multiagentensystem

Für den Begriff Agent im Kontext eines Multiagentensystem liegen viele verschiedene Definitionen vor. Hajduk vergleicht diese unterschiedlichen Definitionen und kreiert folgende eigene Definition (aus dem Englischen übersetzt):

„Ein autonomer Agent ist ein System, das seine Umwelt wahrnimmt und im Laufe der Zeit darauf einwirkt, um seine eigenen Ziele zu verfolgen und die zukünftig wahrgenommene Umwelt zu beeinflussen“ (Hajduk 2019, S. 3)

Diese Definition bildet im Wesentlichen die Schnittmenge anderer Definitionen. Im Zentrum stehen die Eigenschaften:

- autonom
- wahrnehmend
- agierend
- eigene Ziele verfolgend

Die Definition nach Woolridge beschreibt beispielsweise zusätzlich die Fähigkeit zur Interaktion mit anderen Agenten (soziale Fähigkeit). Weitergehend wird bei dieser Definition

in Hardware- und Software-Agenten und darüber hinaus auch in starke und schwache Agenten unterschieden. Andere Definitionen nennen zusätzliche Eigenschaften, wie lernend oder proaktiv. Teilweise wird der Begriff auf Softwaresysteme eingeschränkt. (Hajduk 2019, S. 3)

Die in dieser Arbeit durchzuführende Simulation modelliert Agenten, die in der abgebildeten Realität auch physische Komponenten aufweisen. Aus diesem Grund wird in der vorliegenden Arbeit die Definition von Hajduk verwendet, die Agenten nicht auf Softwaresysteme einschränkt. In Abbildung 3 wird das Zusammenspiel der vier Eigenschaften eines Agenten graphisch dargestellt. Der Status der Umwelt wird durch die Sensoren des Agenten wahrgenommen. Über Aktoren führt er Aktionen aus. Die Entscheidung zu diesen Aktionen wird anhand einer inneren Funktion f des Agenten getroffen. Die Funktion enthält dabei das gesammelte Wissen und die Ziele, nach denen der Agent unter Berücksichtigung des Umweltstatus entscheidet.

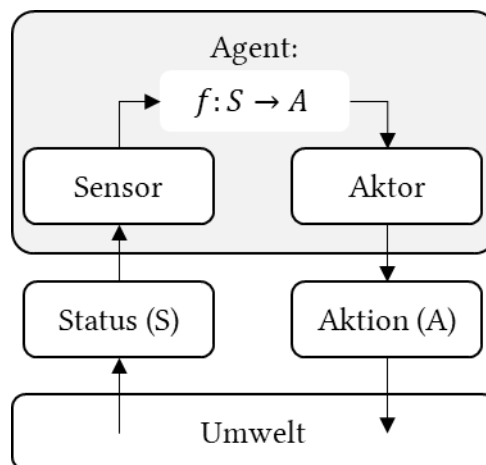


Abbildung 3: Funktionsweise eines Agenten

Agenten lassen sich anhand der Art wie diese Entscheidungsfunktion (bzw. diese Intelligenz) umgesetzt ist unterscheiden. Die einfachsten Realisierungen sind hierbei reaktive Agenten, die anhand von festen Bedingungen festgelegte Aktionen ausführen. Sogenannte kognitive Agenten hingegen sind zur deutlich komplizierteren Verfolgung von Zielvorgaben fähig und können anhand eines inneren Modells der Umwelt auch ihre zukünftigen Aktionen einplanen. (Hajduk 2019, S. 4–8)

Es werden in dieser Arbeit verschieden intelligente Agenten eingesetzt. Ziel ist die Untersuchung der Leistungsfähigkeit von selbstlernenden Agenten. Diese lassen sich dem Bereich der kognitiven Agenten zuordnen. Das *Künstliche Neuronale Netz* wird hierbei eingesetzt, um die Umwelt und die Auswirkungen zukünftiger Aktionen darauf zu modellieren. Die als Leistungsreferenz verwendeten Agenten sind hingegen einfache reaktive Agenten.

Setzt man zur Lösung einer Aufgabe mehrere miteinander in Beziehung stehende Agenten ein, entsteht ein Multiagentensystem. Im Gegensatz zu einem Ein-Agentensystem handelt es sich dabei um eine dezentralisierte Struktur. Das bedeutet, dass die Informationen nicht zentral gesammelt und die Entscheidungen nicht zentral getroffen werden. Stattdessen

entscheidet jede Entität autonom, ist jedoch von den Aktionen der anderen Entitäten zur Zielerfüllung abhängig. (Hajduk 2019, S. 8–9)

Dezentrale Systeme weisen Vorteile beispielsweise in der Skalierbarkeit und der Fehlertoleranz auf (Hajduk 2019, S. 9). Damit eignen sie sich, wie in der Einleitung angeführt, für die interne Flexibilisierung bei ansteigender Komplexität im Umfeld von Unternehmen.

2.3. Selbstlernend - künstliche Intelligenz (KI)

Mit dem Adjektiv ‚selbstlernend‘ werden in dieser Arbeit (in Übereinstimmung mit der Definition des Dudens) Computerprogramme beschrieben, die künstliche Intelligenz besitzen (Dudenredaktion o. J.). Künstliche Intelligenz beschreibt dabei allgemein die maschinelle Nachbildung menschlicher Intelligenz.

Diese Nachbildung von KI kann auf verschiedene Arten umgesetzt werden. Es lassen sich wie in Abbildung 4 dargestellt, anhand von Merkmalen verschiedene Klassen der künstlichen Intelligenz bilden.

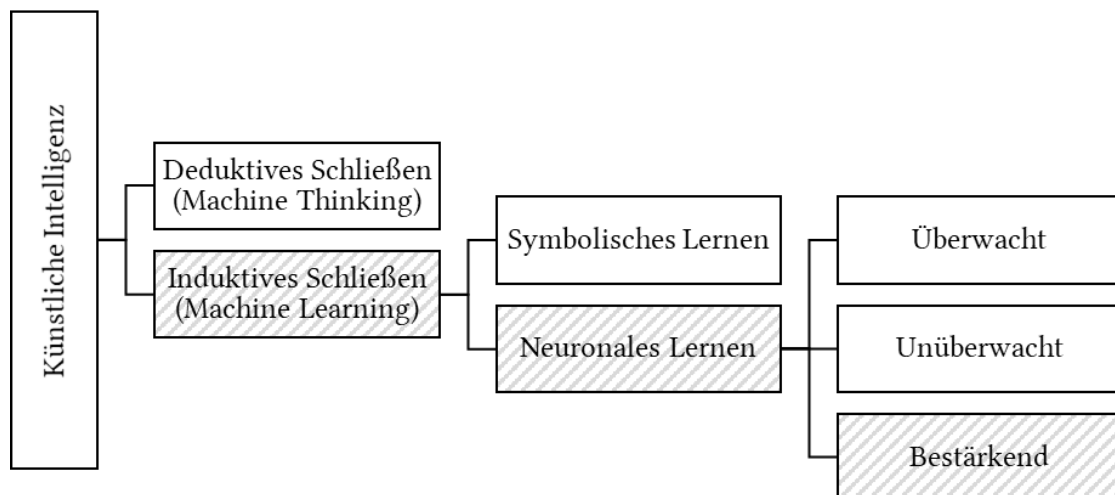


Abbildung 4: Klassifikation Künstlicher Intelligenz

Wie in der Einleitung festgelegt, wird in dieser Arbeit die Intelligenz der Agenten durch bestärkendes Lernen mit einem Künstlichen Neuronalen Netz (KNN) zur Wissensabbildung gebildet. Die entsprechenden Klassen sind in Abbildung 4 hervorgehoben. Diese Auswahl wurde aufgrund der positiven Erfahrungen mit dieser Form der KI in einem Vorgängerprojekt getroffen (Martin Schlump 2020, S. 77–78). Im Vorgängerprojekt wurde ein Ein-Agentensystem zur Problemlösung eingesetzt. Mit dieser Arbeit kann das bestehende selbstlernende Ein-Agentensystem zu einem Multiagentensystem weiterentwickelt werden.

Bei der eingesetzten KI handelt es sich demnach um eine Form des maschinellen Lernens (engl. Machine Learning). Das neu erlernte Wissen wird hierbei selbstständig durch *induktives Schließen* aus Daten gewonnen. Beispiele aus den Daten werden dabei zu Regeln verallgemeinert. Diese Form lässt sich vom *deduktiven Schließen* abgrenzen. Bei diesem wird im Gegensatz dazu aus gegebenen Axiomen durch das Anwenden von Logik neues Wissen erzeugt. (Otte 2019, S. 34)

Neben der Methode, wie das neue Wissen erzeugt wird, ist auch die Form der Wissensrepräsentation ein Merkmal zur genaueren Klassifikation von KI. Das Wissen kann

sowohl symbolisch als auch sub-symbolisch (Synonym: neuronal) abgebildet werden. Bei der sub-symbolischen Abbildung des Wissens geht der Einblick auf die den Entscheidungen zugrundeliegenden Regeln und Daten verloren. Zwar liegt ein eindeutig berechenbares Verhalten vor, allerdings lässt sich dessen Ursprung nicht mehr nachvollziehen (Otte 2019, 105-108). Die in dieser Arbeit angewandte Repräsentation als Künstliches Neuronales Netz ist eine solche sub-symbolische Abbildung des Wissens.

Alternativ zu der eingesetzten bestärkenden Lernmethodik (englisch Reinforcement Learning) können auch sogenannte unüberwachte oder überwachte Methoden eingesetzt werden. Die verschiedenen Lernmethoden ermöglichen die Lösung verschiedener für sie typische Aufgabenstellungen. Das unüberwachte Lernen wird dazu verwendet, die verfügbaren Daten auf Muster zu untersuchen. Beim überwachten Lernen hingegen wird der Algorithmus mit Paaren von Eingangs- und Ausgangswerten trainiert. Ziel ist es, die Funktion, die den Zusammenhang zwischen den Eingangs- und Ausgangswerten beschreibt zu approximieren und damit unbekannte Ausgangsdaten zu ermitteln. Das bestärkende Lernen wird zur Optimierung von Entscheidungsstrategien eingesetzt. Aus diesem Grund eignet sich diese Lernmethodik auch besonders zum Einsatz in der Produktionssteuerung (Kuhnle et al. 2021, S. 856).

Das bestärkende Lernen eignet sich für das Ermitteln von möglichst optimalen Strategien, wenn das Problem als Markow-Entscheidungsprozess (MDP) modelliert werden kann (Kuhnle et al. 2021, S. 857). Die bestimmende Eigenschaft ist hierbei, dass die Wahrscheinlichkeit, einen Zustand s' von Zustand s aus zu erreichen, nur von s und nicht von dessen vorangegangenen Zuständen abhängt.

Das bestärkende Lernen wird fast immer mit agentenbasierten Ansätzen umgesetzt. Das bedeutet, dass die Lernmethode die Funktionsweise eines Agenten, wie in Abbildung 3 dargestellt, nutzt. Der „Reinforcement Learning Agent“ entscheidet sich ausgehend vom wahrgenommen Status s für eine Aktion a . Diese Aktion führt zu einer Veränderung der Umgebung in Status s' . Ist diese Veränderung wünschenswert, erhält der Agent eine Belohnung oder andernfalls eine Bestrafung. Anhand dieser Rückmeldungen lernt der Algorithmus bei häufiger Durchführung, welche Aktion im jeweils aktuellen Status voraussichtlich zu einem wünschenswerten Folgezustand führt (Frochte 2019, S. 23–24). Hierfür bildet das Lernverfahren beim sogenannten Q-Learning Zustands-Aktions-Paare $Q(s_t, a_t)$, die den Wert der Aktionen in diesem Status beschreiben. Die Entscheidungsfunktion f , d.h. die Strategie des Agenten wird darauf aufbauend angepasst, um die Belohnung langfristig zu maximieren.

Mit dem bestärkenden Lernen ist sogar bei einer unendlichen Anzahl an möglichen Status und Aktionen die Annäherung an eine optimale Strategie möglich. Dabei ist jedoch eine ausreichende Form der Wissensspeicherung notwendig. In dieser Arbeit wird die Strategie, wie zuvor ausgeführt, in einem neuronalen Netz gespeichert, das diese Bedingung erfüllt. Beim sogenannten Deep Q-Learning werden die Informationen über den Status als Eingangsgröße an das neuronale Netz übergeben und dieses gibt einen Vektor mit Q-Werten aller möglichen Aktionen als Ausgangsgröße. Die neuronalen Netze werden in diesem Zusammenhang auch als Deep Q-Network (DQN) bezeichnet.

Die in dieser Arbeit eingesetzten selbstlernenden Agenten lernen auf diese Weise ihre eigene Heuristik für die simulierte Produktionssteuerung. Diese Heuristik kann anschließend angewendet werden, indem die durch den Algorithmus getroffenen Entscheidungen

ausgeführt werden. Teilweise kann es bei bestehender Heuristik auch sinnvoll sein, die Produktionssteuerung an Agenten mit deaktivierter Lernfähigkeit zu übergeben. Auf diese Weise wird die sonst inhärente Exploration mittels zufälliger Entscheidungen vermieden.

Auch bei deaktivierter Lernfähigkeit ist die Bezeichnung als selbstlernendes System weiterhin zulässig. So ist die angewandte Heuristik durch einen selbstlernenden Algorithmus entstanden und kann durch einfaches Aktivieren der Lernfähigkeit weiter trainiert werden.

2.4. Simulation

Die Simulation als Problemlösungsmethode baut auf der System- und Modelltheorie auf. Ein reales oder geplantes System wird in einem Modell abgebildet, um es anschließend simulationsunterstützt zu untersuchen (Gutenschwager et al. 2017, S. 10). Die zugrundeliegenden Begriffe sind in der VDI-Richtlinie 3633 Blatt1 definiert und werden entsprechend im Folgenden in starker Anlehnung verwendet (VDI-Richtlinien 3633, S. 3; Gutenschwager et al. 2017):

- System: eine von der Umwelt abgegrenzte Menge von untereinander in Beziehung stehenden Elementen
- Modell: vereinfachte Nachbildung eines Systems mit seinen Prozessen
- Simulation: Nachbildung eines Systems mit seinen dynamischen Prozessen in einem experimentierbaren Modell, um zu Erkenntnissen zu gelangen, die auf die Wirklichkeit übertragbar sind

Der Einsatz von Simulation dient entsprechend der verwendeten Definition immer dem Erkenntnisgewinn über das simulierte System. Diese Erkenntnisse werden durch Experimentieren mit dem Simulationsmodell gewonnen. Dies bedeutet, dass gezielte empirische Untersuchungen des Modellverhaltens durch wiederholte Simulationsläufe durchgeführt werden. Hierbei werden Parameter und Struktur des Modells systematisch variiert (VDI-Richtlinien 3633, S. 3)

Innerhalb dieser Arbeit wird der Begriff auf die Simulation mittels Computer eingeschränkt. Dies bedeutet, dass das Simulationsmodell inklusive der abgebildeten Elemente als Computerprogramm ausgeführt wird (Gutenschwager et al. 2017, S. 22). Als Simulationslauf werden die zum Experimentieren durchgeführten Ausführungen dieses Programms bezeichnet.

Wie in der Einführung erläutert, baut diese Arbeit auf einer bestehenden Masterarbeit an der Technischen Hochschule Ulm auf. Bei dieser Vorläuferarbeit ist die Computersimulation mittels der Softwarewerkzeuge Plant Simulation und Python umgesetzt worden (Martin Schlump 2020, S. 48). Diese Kombination wurde deshalb ebenfalls verwendet.

In Abgrenzung zu statischen Analysen bildet die Simulation auch die dynamischen Systemveränderungen ab. Die Abbildung des Zeitverhaltens kann auf unterschiedliche Weise erfolgen. Unterschieden wird im Allgemeinen in kontinuierliche und diskrete Simulation. Die Simulation basiert entweder auf einer kontinuierlichen oder diskreten Zeitmenge. Für die Simulation von Produktionssteuerung ist die diskrete Simulation üblich, weil sich der Zustand des Systems nur beim Eintreten eines Ereignisses, also zu bestimmten Zeitpunkten verändert. (Gutenschwager et al. 2017, S. 24) Bei der angewandten Simulationsmethode handelt es sich demnach um eine ereignisgesteuerte, diskrete, computergestützte Ablaufsimulation.

3. Konzeption und Umsetzung des Simulationsmodells

Zur Untersuchung der Leistungsfähigkeit des Multiagentensystems bedarf es einem simulierbaren Modell eines Produktionssystems. Die Anforderungen an dieses Modell lassen sich im Wesentlichen in den folgenden drei Gruppen zusammenfassen:

- Realitätsnähe
- Verständlichkeit
- Komplexität

Realitätsnähe: Durch Realitätsnähe des Simulationsmodells sind die gewonnen Erkenntnisse besser auf die Realität übertragbar. Das Modell sollte demnach ein Produktionssystem abbilden, dass in dieser Form auch tatsächlich sinnvoll eingesetzt werden könnte.

Verständlichkeit: Das Modell sollte möglichst einfach verständlich sein. Auf diese Weise können die Erkenntnisse zielgerichtet vermittelt werden. Die Konzentration kann auf die Untersuchungen am Modell gerichtet werden und ein hoher Aufwand an Einarbeitung in das Modell entfällt.

Komplexität: Um ein selbstlernendes Multiagentensystem sinnvoll einzusetzen, bedarf es einer ausreichend komplexen Problemstellung. Das modellierte Produktionssystem muss sich für den Einsatz von mindestens drei Agenten eignen. Außerdem darf das Problem über keine einfach zu ermittelnde optimale Lösung verfügen.

Die Forderungen nach ausreichender Komplexität bei gleichzeitig einfacher Verständlichkeit stehen in einem Spannungsverhältnis. Innerhalb des folgenden Kapitels gilt es, ein Simulationsmodell zu entwickeln, das allen Anforderungsbereichen möglichst gerecht wird.

Hierzu werden zuerst Beispiele aus der Literatur beschrieben, die verschiedene Ansätze aufweisen und als Inspirationsquelle dienen. Anschließend wird das fiktive Produktionssystem entwickelt und die Gestaltung des Multiagentensystems entsprechend festgelegt. Weitergehend werden diese Systeme mittels der bestehenden Softwarebausteine in Plant Simulation und Python aufgebaut. Insbesondere werden dabei die notwendigen Anpassungen an den Bestandteilen der Vorgängerarbeit beschrieben. Das resultierende Simulationsmodell ist online auf GitHub einsehbar (Zipfel 2022).

3.1. Beispiele aus der Literatur

Vor der Festlegung des fiktiven Produktionssystems und dessen Modellierung für diese Arbeit wurde eine Literaturrecherche durchgeführt. Hierbei wurden Beispiele gesucht in denen selbstlernende Agentensysteme vergleichbare Steuerungsaufgaben übernehmen. Mit vergleichbar ist in diesem Zusammenhang gemeint, dass ähnliche Problemstellungen wie in der Produktionssteuerung zu lösen sind.

In den beiden folgenden Tabellen werden vier solcher Literaturbeispiele vorgestellt. Zwei davon sind Einzelagentensysteme und zwei Multiagentensysteme.

Bei allen Systemen haben die Agenten übereinstimmend einen sehr reduzierten Aktionsraum zur Verfügung. Außerdem sind die Agenten in allen Fällen so gestaltet, dass sie im

Produktionssystem einem logischen Subsystem zuzuordnen sind (Ladungsträger, Fahrstühle oder einzelne Produktinstanzen).

Die Form des Selbstlernens ist in drei der vier Beispiele das Q-Learning. Allerdings wird bei Zweien auf eine Wissensrepräsentation mittels neuronalen Netzes verzichtet und eine Matrix eingesetzt. Das vierte Beispiel setzt neuronales Lernen ohne Q-Learning ein.

In beiden Beispielen mit Multiagentensystem wird hervorgehoben, dass die Belohnungsfunktion für das Selbstlernen zumindest eine globale Komponente enthält. Alle Agenten im System werden damit zumindest teilweise am Gesamterfolg gemessen.

Titel	A reinforcement learning based approach for a multiple-load carrier scheduling problem (Chen et al. 2015)	Elevator Group Control Using Multiple Reinforcement Learning Agents (Crites und Barto 1998)
Systembeschreibung	Mehrfach-Ladungsträger (=Agent) liefert Teile zur Produktion an fließbandseitige Pufferspeicher.	Vier Fahrstühle (= Agenten) in unterschiedlichen Schächten transportieren Fahrgäste.
Ziel	Durchsatz erhöhen	durchschnittliche quadratische Wartezeit minimieren
Anzahl der Agenten	Einzelagent-System	Multi-Agenten-System
Aktionsraum	Teile aufnehmen und liefern (Auswahl eines von zwei vorprogrammierten Regelwerken zur Ermittlung der Reihenfolge); Warten	wenn stehend: Hoch- oder Herunterfahren; wenn fahrend: Stoppen oder Durchfahren
Form des Selbstlernens	Q-learning (Matrix)	Q-learning (Matrix)
Zusätzliche Hinweise		globale Belohnungsfunktion

Tabelle 1: selbstlernende Agentensysteme, Literaturrecherche Teil A

Titel	Designing an adaptive production control system using reinforcement learning (Kuhnle et al. 2021)	Autonomous production control for matrix production based on deep Q-learning (Hofmann et al. 2020)
Systembeschreibung	Ladungsträger (=Agent) liefert Teile in Eingangspuffer der Maschinen und holt diese aus den Ausgangspuffern ab.	Matrixproduktion mit mehreren Produktfamilien und -varianten; Produktinstanz = Agent
Ziel		Minimieren der Durchlaufzeit
Anzahl der Agenten	Einzelagent-System	Multi-Agenten-System
Aktionsraum	Transport; zum Abholen Fahren; Warten	Maschine und gewünschte Bearbeitungsprozesse wählen; Warten
Form des Selbstlernens	„Different Policy-gradient methods“ (bspw. TRPO – neuronales Netz)	Deep-Q-Learning (neuronales Netz)
Zusätzliche Hinweise		Zweigeteilte Belohnungsfunktion, d.h. nach jeder Aktion und nach Fertigstellung

Tabelle 2: selbstlernende Agentensysteme, Literaturrecherche Teil B

Zusammenfassend lässt sich festhalten, dass die Möglichkeiten zur Gestaltung der Systeme sehr vielfältig sind. Eine dominante Strategie in der Konzeption von selbstlernenden Agentensystemen liegt nicht vor. Die Gestaltung wird sehr individuell auf die jeweils vorliegende Problemstellung ausgerichtet.

3.2. Gestaltung des fiktiven, physischen Produktionssystems

Das für diese Arbeit verwendete fiktive Produktionssystem ist in Abbildung 5 visualisiert. Das Produktionssystem besteht aus den folgenden physischen Bestandteilen:

- Maschine (engl. Machine)
- Eingangspuffer (engl. BufferIn)
- Förderband (engl. Conveyor)
- Quelle Ladungsträger (engl. Source Carrier)
- Ladungsträger (engl. Carrier)
- Quelle (engl. Source)
- Auftrag (engl. Order)
- Senke (engl. Drain)

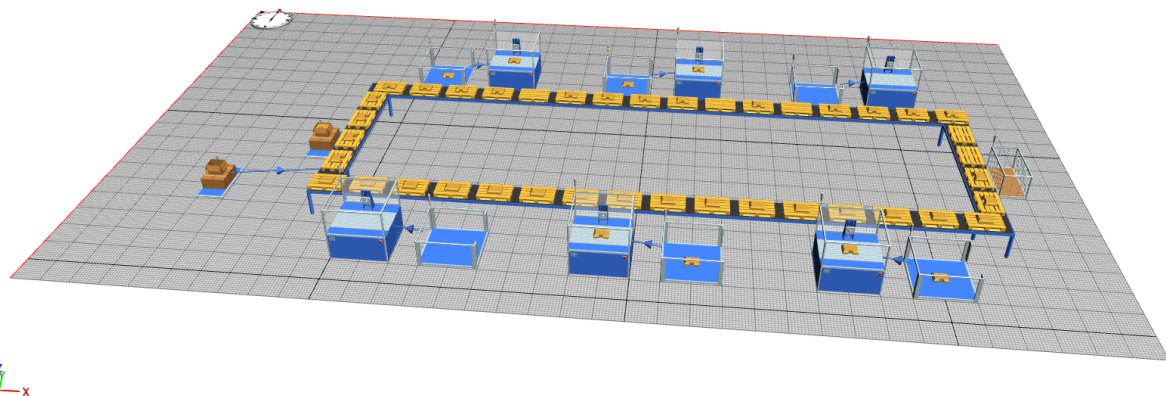
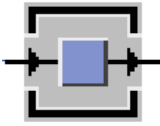
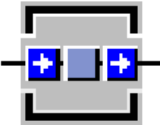

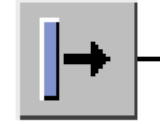

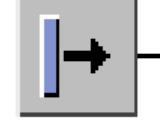



Abbildung 5: 3D-Darstellung des Produktionssystems, modelliert in Plant Simulation

Die grundlegende Funktionsweise lässt sich folgendermaßen beschreiben: Die sechs Maschinen mit zugehörigem Eingangspuffer sind über das Förderband untereinander und mit der Quelle und der Senke verbunden. Das Förderband transportiert die Ladungsträger im Kreis. Auf den Ladungsträgern werden die ursprünglich von der Quelle ausgegebenen Produktionsaufträge fortbewegt. Ist ein Auftrag vollständig durch die Maschinen bearbeitet wird dieser an die Senke abgegeben.

Alle physischen Bestandteile sind mit Instanzen von angepassten Standardklassen aus Plant Simulation modelliert. In Tabelle 3 sind deren Funktionen und Attribute detailliert beschrieben.

Klasse: 2D-Grafik	Beschreibung der Funktionen und Attribute
Maschine: 	<p>Die Instanzen der Klasse Maschine verfügen über verschiedene Fähigkeiten zur Bearbeitung von Aufträgen. Diese Fähigkeiten sind zentral definiert und den verschiedenen Instanzen zugeordnet. In der aktuellen Konfiguration gibt es beispielsweise fünf verschiedene Bearbeitungsfähigkeiten, von denen jede Instanz über nur drei verfügt. Wird ein Auftrag von einer Maschine aufgenommen, werden alle direkt anstehenden Bearbeitungsschritte durchgeführt, zu denen die Maschine befähigt ist. Ist die Bearbeitung abgeschlossen, wird der Auftrag an den nächsten verfügbaren freien Ladungsträger abgegeben. Vor der Bearbeitung des nächsten Auftrags bedarf es einer Erholzeit, die aktuell bei 15min liegt.</p>
Eingangspuffer: 	<p>Die Instanzen der Klasse Eingangspuffer sind jeweils einer Maschine vorgelagert. Aufträge werden vom Förderband an den Eingangspuffer abgegeben. Dieser gibt den Auftrag an die Maschine weiter, wenn diese frei ist. Die aktuelle Kapazität aller Eingangspuffer beträgt eins.</p>
Förderband: 	<p>Das Förderband verbindet die Objekte des Produktionssystems in einem kontinuierlichen Kreislauf. Stellen, an denen die Abgabe oder Aufnahme von Aufträgen erlaubt ist, sind durch Sensoren gekennzeichnet. Jeder dieser Sensoren ist einem entsprechenden Objekt zugeordnet. Das Förderband nimmt in der aktuellen Konfiguration 40 Ladungsträger zur Beförderung der Aufträge auf.</p>
Quelle Ladungsträger: 	<p>Die Quelle Ladungsträger befüllt das Förderband zu Beginn eines Simulationslaufs mit einer festen Anzahl an Ladungsträgern.</p>
Ladungsträger: 	<p>Die Instanzen der Klasse Ladungsträger dienen dem Transport der Aufträge auf dem Förderband. Die Ladungsträger verlassen das Förderband während des Betriebs nicht. Jede Instanz verfügt aktuell über die Kapazität eines Auftrags.</p>
Quelle: 	<p>Die Quelle generiert in relativ kurzen, zufallsverteilten Zeitabständen neue Aufträge. Die Aufträge werden an den nächsten verfügbaren freien Ladungsträger abgegeben, wenn eine feste Anzahl an Aufträgen im System nicht überschritten ist. Aktuell liegt diese Grenze bei 43 Aufträgen.</p>
Auftrag: 	<p>Jede Instanz der Klasse Auftrag verfügt über eine individuelle Laufkarte (engl. route sheet). Diese Laufkarte beschreibt, wie der Auftrag zu bearbeiten ist (Arbeitsschritte, Bearbeitungsdauer und Reihenfolge). Darüber hinaus wird darin der Status der Bearbeitung dokumentiert.</p> <p>Die Laufkarte wird bei der Erstellung des Auftrags zufallsgeneriert. Die Anzahl der Bearbeitungsschritte ist dabei aktuell normalverteilt im Intervall {4; ...; 6}. Die Art des jeweiligen Arbeitsschritts ist gleichverteilt über die fünf verschiedenen Möglichkeiten und die jeweilige Bearbeitungszeit ist Erlang-verteilt zwischen 5 und 30 min ($\mu = 10$; $\sigma = 5$).</p>


Klasse: 2D-Grafik	Beschreibung der Funktionen und Attribute
Senke: 	Die Senke nimmt fertiggestellte Aufträge auf, wenn diese den entsprechenden Sensor passieren.

Tabelle 3: Detailbeschreibung der zur Modellierung verwendeten Klassen

Das vorgestellte fiktive Produktionssystem wurde unter den zu Beginn des Kapitels vorgestellten Anforderungen gestaltet und weiterentwickelt.

Für eine gute *Realitätsnähe* wurden Objekte verwendet, die ohne hohen Abstraktionsaufwand als Objekte aus realen Produktionsszenarien verstanden werden können. Die Maschinen lassen sich beispielsweise als Bearbeitungszentren mit verschiedenen verfügbaren Werkzeugen interpretieren. Die Aufträge können als Werkstücke mit den zugehörigen Informationen zu deren Bearbeitung verstanden werden. Quelle und Senke stellen die Übergabepunkte zu vorgelagerten bzw. nachgelagerten Abteilungen dar.

Das Förderband bildet den physischen Transport der Werkstücke sichtbar und realistisch ab. Der Vorgang zur Aufnahme und Abgabe der Werkstücke an den entsprechenden Stellen des Förderbands ist jedoch nicht modelliert. Diese Vereinfachung lässt sich als realitätsfern kritisieren. Eine diesbezüglich weitergehende Detaillierung hätte durch zusätzliche Objekte und Steuerungsbedarf jedoch die Verständlichkeit des Modells reduziert. Außerdem lässt sich davon ausgehen, dass die Vereinfachung keine Auswirkungen auf die grundsätzlichen Ergebnisse der geplanten Simulationen hat.

Durch die geringe Anzahl an unterschiedlichen Objekten konnte eine hohe intuitive *Verständlichkeit* der Vorgänge im Modell erreicht werden. Alle Objekte können bei der 2D oder 3D Visualisierung in Plant Simulation einfach eindeutig identifiziert werden. Die Reduktion auf einen einzigen Zugangspunkt (Quelle) und einen einzigen Ausgangspunkt (Senke) ermöglicht eine übersichtliche Einschätzung der Gesamtleistung.

Die für die Untersuchung der Fragestellung notwendige *Komplexität* wird durch die erzwungene Zusammenarbeit der Maschinen untereinander erreicht. Weil eine einzelne Maschine nur über eine Teilmenge der notwendigen Fähigkeiten verfügt, ist ein Zusammenwirken notwendig, um Aufträge abzuschließen.

Hinzu kommen weitere Möglichkeit zur Steigerung der Komplexität des Produktionssystems. Beispielsweise ist die Erweiterung oder Reduktion der Maschinenanzahl einfach umsetzbar. Die notwendigen Bearbeitungsfähigkeiten können neu zugeteilt, erweitert und angepasst werden. Auch die Logik zur zufälligen Erstellung der Auftragslaufkarten lässt sich zur Komplexitätssteigerung ohne großen Aufwand verändern.

3.3. Gestaltung des Multiagentensystems

Die Steuerung des fiktiven Produktionssystems erfolgt entsprechend der Zieldefinition dieser Arbeit mittels eines dezentralen Agentensystems. In Abbildung 6 ist dargestellt, wie dieses umgesetzt ist. Jedem Maschine-Eingangspuffer-Paar ist ein Agent zur Steuerung zugeordnet. Somit besteht, wie in den Literaturbeispielen, die direkte Zuordnung zwischen einem wahrgenommenen, logischen Subsystem und einem Agenten.

Ein weiterer Teil der Steuerungslogik auf Gesamtsystemebene ist die Einrichtung der Schnittstelle zwischen den beiden Softwarewerkzeugen Python und Plant Simulation. Die bidirektionale Verbindung wird mittels Standardklassen unter der Nutzung des TCP/IP Netzwerkprotokolls aufgebaut. Die Socket-Schnittstelle muss vor Benutzung zur Simulation zentral gestartet werden.

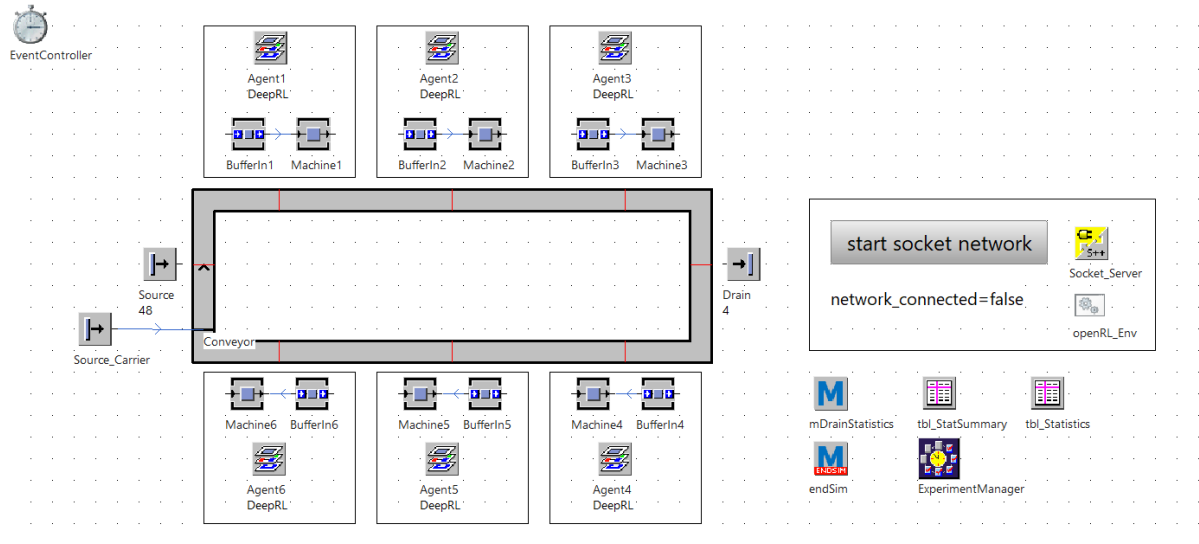


Abbildung 6: 2D-Darstellung des Produktionssystems inkl. Objekte zur Steuerung, modelliert in Plant Simulation

Die Steuerung der Agenten kann durch verschiedene Heuristiken erfolgen. In Abbildung 7 sind die aktuell im Modell verfügbaren Heuristiken aufgelistet. Die Heuristiken `nextValidAction` (nächste zulässige Aktion) und `shortestRemainingTime` (kürzeste Restbearbeitungszeit) dienen dem Leistungsvergleich. Mit den Ergebnissen dieser Steuerungsstrategien können die Ergebnisse des selbstlernenden Ansatzes (DeepRL) verglichen werden.

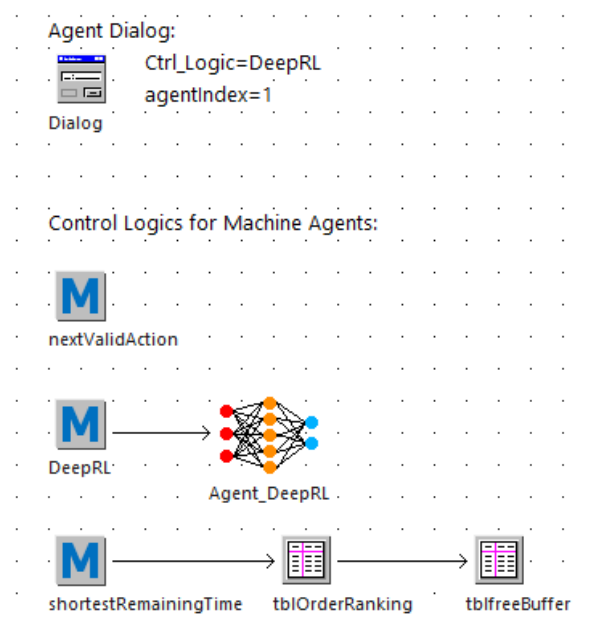


Abbildung 7: Verfügbare Heuristiken zur Steuerung der Agenten

Bei den Vergleichsheuristiken handelt es sich um statische Steuerungslogiken. Die Entscheidungen werden immer entsprechend den zuvor programmierten festen Grundsätzen getroffen. Im Gegensatz dazu verändert sich die Steuerungslogik DeepRL, wenn die Lernfähigkeit aktiviert ist. Dies hat zur Folge, dass Entscheidungen bei gleicher Ausgangslage zu unterschiedlichen Zeitpunkten unterschiedlich ausfallen können. Ist die Lernfähigkeit deaktiviert, verhält sich auch diese Heuristik statisch.

Auch bei der Gestaltung des Multiagentensystems wurden die zu Beginn des Kapitels vorgestellten Anforderungen berücksichtigt.

Die einfache *Verständlichkeit* wird durch die direkte Zuordnung zwischen einem logischen Subsystem und einem Agenten unterstützt. Auch bei einer Erweiterung des Produktionssystems unterstützt diese Zuordnung die Übersichtlichkeit. Die Festlegung deckt sich mit den als agierende Einheiten wahrgenommen Subsystemen.

Durch die räumliche Trennung kann transparent nachvollzogen werden, welche Entscheidungen von welchem Agenten oder anderweitig getroffen werden. Dies unterstützt die Verständlichkeit beim Nachvollziehen getroffener Entscheidung während einer Simulation.

Das Produktionssystem erweist sich ferner als ausreichend *komplex*. Die Größe macht eine vollständige Statusbeobachtung unpraktikabel und zwingt daher zu einer unvollständigen Beobachtung. Diese selektive Beobachtung ist auch bei realen Produktionssystem notwendig.

3.4. Heuristiken zur Produktionssteuerung

Alle Agenten weisen, wie mit Abbildung 3 eingeführt, mehrere Bestandteile auf. Kern ist die Entscheidungsfunktion, die im aktuellen Simulationsmodell einer der drei Heuristiken entspricht. Vor der Durchführung von Simulationsläufen ist es notwendig, die gewünschten Steuerungslogiken für die Agenten festzulegen. Abbildung 8 beschreibt das notwendige Vorgehen hierfür.

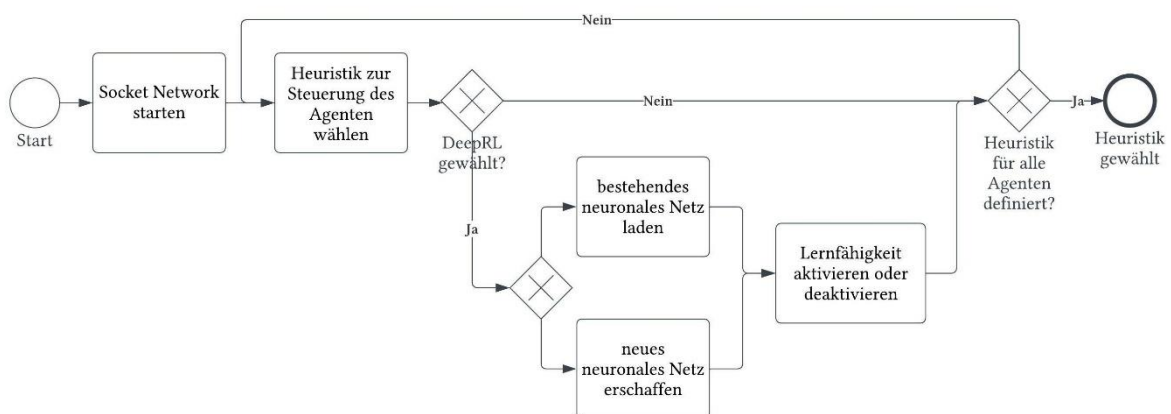


Abbildung 8: Auswahl der Steuerungslogik

Als Entscheidungsgrundlage bedarf es zur Anwendung der Heuristiken den beobachteten Status der Umwelt. Je nach Entscheidungslogik sind unterschiedliche Beobachtungen notwendig. Diese Teilmenge an Statusbeobachtungen wird im Folgenden als

Beobachtungsraum (engl. observation space) bezeichnet. Der Beobachtungsraum ist für die jeweilige Heuristik entsprechend anzupassen.

Entschieden wird über die Ausführung der verfügbaren Aktionen. Diese Menge an Aktionen wird im Folgenden Aktionsraum (engl. action space) genannt. Dieser Aktionsraum ist von der Steuerungslogik unabhängig. Ein Agent verfügt aktuell immer über die drei Aktionsmöglichkeiten:

1. Umlagern eines Auftrags vom Ladungsträger auf den Eingangspuffer
2. Umlagern eines Auftrags von der Maschine auf den Ladungsträger
3. Keine Umlagerung vornehmen

Der Entscheidungsprozess wird in allen Fällen durch das Passieren eines Ladungsträgers des entsprechenden Sensors auf dem Förderband ausgelöst.

nextValidAction (nächste zulässige Aktion)

Diese Heuristik stellt die einfachste aktuell eingesetzte Steuerungslogik dar. Passiert ein Ladungsträger den einem Agenten zugewiesenen Sensor, stellt dieser die folgenden Beobachtungen an:

1. Ist der Ladungsträger mit einem Auftrag belegt?
2. Ist der Eingangspuffer belegt?
3. Kann der nächste notwendige Bearbeitungsschritt durch die Maschine durchgeführt werden?
4. Ist die Maschine mit einem Auftrag belegt und hat alle möglichen Bearbeitungen durchgeführt?

Basierend auf diesen Beobachtungen wird ermittelt, ob der Transfer eines Auftrags möglich ist. Dies ist der Fall, wenn der Ladungsträger unbelegt und die Bearbeitung an einem Auftrag abgeschlossen ist. Oder wenn der Eingangspuffer frei ist und der passierende Auftrag bearbeitet werden kann. Ist ein Transfer möglich, wird dieser immer durchgeführt. Ansonsten wird kein Transfer durchgeführt.

shortestRemainingTime (kürzeste Restbearbeitungszeit)

Diese Heuristik weist eine höhere Leistungsfähigkeit als die einfachere nextValidAction auf. Passiert ein Ladungsträger, stellt der Agent die folgenden Beobachtungen an:

1. Ist der Ladungsträger mit einem Auftrag belegt?
2. Ist der Eingangspuffer belegt?
3. Ist die Maschine mit einem Auftrag belegt und hat alle möglichen Bearbeitungen durchgeführt?
4. Ist die Maschine die beste freie Wahl (die meisten aufeinanderfolgenden möglichen Bearbeitungsschritte) für einen durch die kürzeste Restbearbeitungszeit priorisierten (je kürzer desto höher die Priorität) Auftrag?

Ist der Transfer eines fertig bearbeiteten Auftrags auf den leeren Ladungsträger möglich, wird dieser immer durchgeführt. Ein zu bearbeitender Auftrag wird nur in den Eingangspuffer transferiert, wenn dieser priorisiert ist und die Maschine die beste Wahl zur Bearbeitung ist. In allen anderen Fällen wird kein Transfer durchgeführt.

DeepRL (selbstlernende Heuristik)

Für die selbstlernende Heuristik ist der Beobachtungsraum am größten:

1. Ist der Ladungsträger mit einem Auftrag belegt?
2. Ist der Eingangspuffer belegt?
3. Ist die Maschine mit einem Auftrag belegt?
4. Wie viele belegte Ladungsträger befinden sich auf dem Förderband?
5. Wie viele Aufträge befinden sich insgesamt im Produktionssystem?
6. Wie viele aufeinander folgende Bearbeitungsschritte können am passierenden Auftrag ausgeführt werden?
7. Hat die Maschine die derzeitige Bearbeitung abgeschlossen?
8. Wie viele Bearbeitungsschritte sind am passierenden Auftrag noch durchzuführen?
9. Wie viele aufeinander folgende Bearbeitungsschritte können am Auftrag auf dem nachfolgenden Ladungsträger ausgeführt werden?
10. Wie viele aufeinander folgende Bearbeitungsschritte kann die passendste Maschine am passierenden Auftrag ausführen?

In Abbildung 9 ist dargestellt, wie die Beobachtungen für die selbstlernende Heuristik verarbeitet werden. Der Ablauf unterscheidet sich bei aktivierter bzw. deaktivierter Lernfähigkeit.

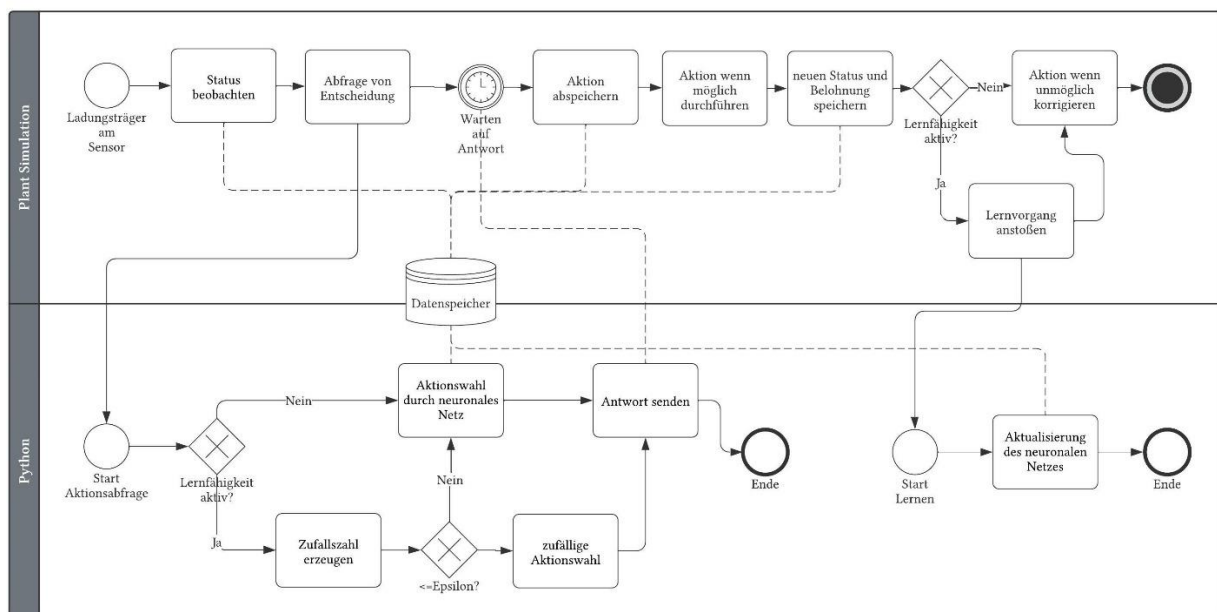


Abbildung 9: Prozessablauf der selbstlernenden Heuristik

Wählt die in Python implementierte Entscheidungsfunktion eine unmögliche Aktion (bspw. Transfer auf einen bereits besetzten Ladungsträger), wird eine negative Belohnung vergeben und diese im Anschluss korrigiert. Dieser Eingriff ist notwendig, um eine Verklemmungssituation zu verhindern und den Simulationslauf weiter aufrecht zu halten. Die Korrektur erfolgt mittels einer zufällig gewählten möglichen Aktion.

Die Aktualisierung der neuronalen Netze bei aktivierter Lernfähigkeit ist maßgeblich durch die Belohnungsfunktion beeinflusst. Beispielhaft ist die in Modell A19b und A35 umgesetzte Belohnungsfunktion in Abbildung 10 dargestellt (s. zur Modellübersicht Abbildung 14). Eine detaillierte Beschreibung des Lernprozesses (Aktualisierung des neuronalen Netzes) findet sich in der Vorgängerarbeit (Martin Schlump 2020, S. 46).

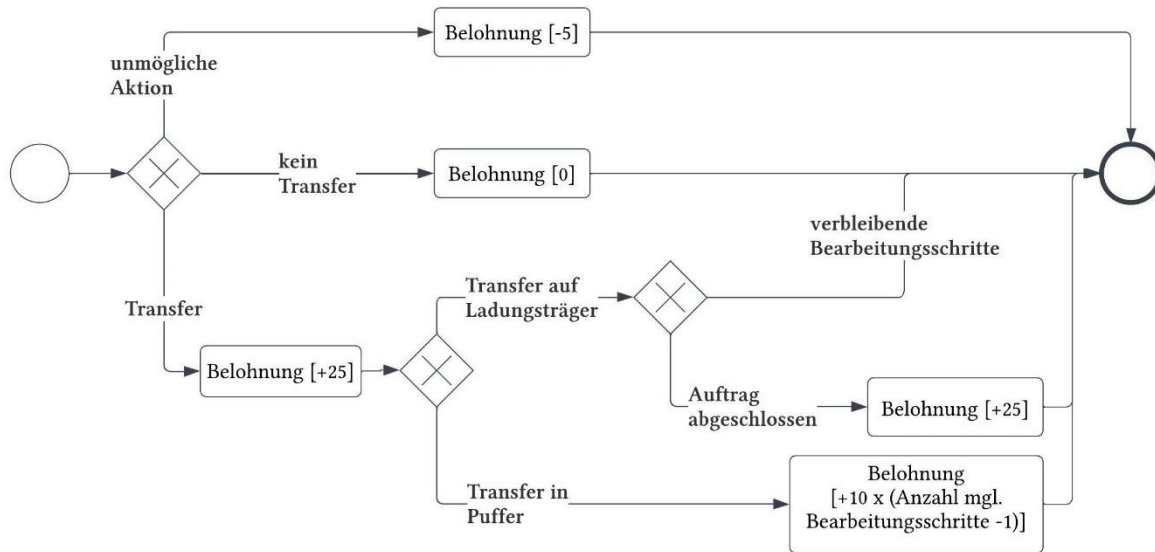


Abbildung 10: Belohnungsfunktion der Modellversionen A19b und A35

3.5. Anpassung bestehender Softwarekomponenten

Wie bereits vorgestellt, basiert die Umsetzung der selbstlernenden Agenten auf Bausteinen einer Vorgängerarbeit (Martin Schlump 2020). In der Vorgängerarbeit wurde ein Ein-Agentensystem modelliert. Die Bausteine mussten zur Verwendung in einem Multiagentensystem angepasst werden. Weitergehend waren Anpassungen zur Verwendung in einem anderen fiktiven Produktionssystem notwendig. Im Folgenden werden wichtige Anpassungen erläutert.

Die Python-Skripte dienen als Lernumgebung für das bestärkende Lernen und verwalten die neuronalen Netze zur Wissensspeicherung der Agenten (siehe hierzu auch Abbildung 9). Diese Skripte wurden zur Verwendung in einem Multiagentensystem angepasst.

Hierzu verwaltet ein Python Dictionary alle Instanzen der Klasse „Agent“. Diese Klasse enthält die Funktionen für das Training und die Entscheidungsfindung. Über den agentIndex kann auf die spezifische Instanz zugegriffen werden. Auf diese Weise können die empfangenen Daten bzw. Aufgaben (task 1 bis 8) korrekt zugeordnet werden (siehe Abbildung 11). Eine detailliertere Beschreibung der Python Skripte findet sich in der Vorgängerarbeit (Martin Schlump 2020, S. 62–68).

```

# Endlosschleife zum Datenempfang, Datenverarbeitung und erledigen der Aufgaben
while True:
    # sende eine Nachricht an PlantSim bei der Initialisierung
    if init == True:...

    # Socket auf Empfang, wartend auf Dateneingänge
    PSinput = client_socket.recv(1024)

    # Teste die Verbindung bevor Daten gesendet/empfangen werden
    if len(PSinput) <= 0:
        break

    # erzeuge ein Objekt "convert" von der Klasse ConvData
    convert = ConvData(PSinput)
    # extrahiere den Index des aufrufenden Agenten
    agentIndex = convert.get_agentIndex(PSinput)
    # wähle zugehörigen agent aus dem dictionary
    agent = agents.get(agentIndex)
    # extrahiere die Aufgabe aus dem PlantSim Input
    task = convert.get_task(PSinput)

    ##### initialisiere den Agenten #####
    if task == 1:...
    #####

    ### speichere Daten in Erfahrungsspeicher und trainiere Netz ##
    elif task == 2:...
    #####

    ##### hole Aktion (Zufall/Vorhersage) von Agent #####
    elif task == 3:...
    #####

    ##### hole Aktion (Vorhersage) vom geadenen Netz #####
    elif task == 4:...
    #####

    ##### sichere Agentennetz (model) #####
    elif task == 5:...
    #####

    ##### lade Agentennetz (model) #####
    elif task == 6:...
    #####

    ### beende Socket-Client Verbindung zum Simulationsagenten ###
    elif task == 8:
        break
    #####

```

Abbildung 11: agentsScript.py

Das Datentransferprotokoll (Martin Schlump 2020, S. 61) wurde zu diesem Zweck um den agentIndex erweitert. Der Index des Agenten in Python entspricht dem Index in Plant Simulation.

Das in Abbildung 7 zu sehende Objekt der Klasse „Agent_DeepRL“ basiert auf der Klasse „Agent“ der Vorgängerarbeit. Dieser Bestandteil steuert wie in Abbildung 9 beschrieben das Verhalten des Agenten innerhalb des Simulationsmodells in Plant Simulation. Darüber hinaus lassen sich über die Objekte dieser Klasse die Eigenschaften der jeweiligen Agenten definieren. In Abbildung 12 sind die Methoden und Variablen der aktuellen Klasse abgebildet.

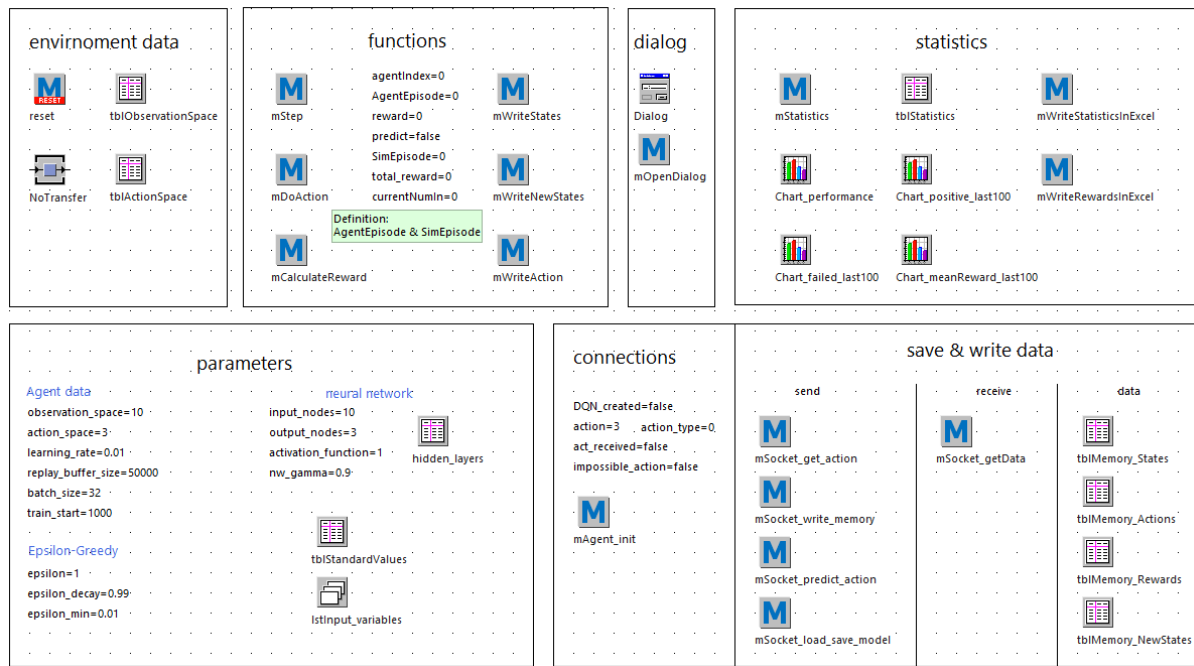


Abbildung 12: Agent_DeepRL (A39)

Die in 3.3 vorgestellte Schnittstelle wird in der aktuellen Form, wie erläutert, auf der Gesamtsystemebene verwaltet. Auf diese Weise können alle Agenten über eine zentrale Schnittstelle kommunizieren. Zuvor war diese innerhalb der Agenten-Klasse definiert.

Die Funktion zur Statusbeobachtung ist an das fiktive Produktionssystem angepasst und beobachtet die zehn definierten Eigenschaften. Ebenfalls ist der Aktionsraum auf die drei verfügbaren Aktionen eingestellt. Die Parameter für das neuronale Netz sind ebenfalls entsprechend eingestellt (10 Eingangsneuronen und 3 Ausgangsneuronen).

Eine weitere Anpassung ist der Umgang mit Entscheidungen zu unmöglichen Aktionen. Diese werden nach der Vergabe einer negativen Belohnung vor der Ausführung korrigiert. Es wird stattdessen eine zufällige, mögliche Aktion ausgeführt. Dieses Vorgehen ermöglicht einen verklemmungsfreien Simulationslauf über 7 Tage Produktion. Durch Hochzählen der Variable SimEpisode wird jedoch sichergestellt, dass Belohnungen nicht über eine solche Fehlentscheidung hinweg diskontiert werden. Für das Training der neuronalen Netze werden Belohnungen nur über Ketten möglicher Entscheidungen hinweg zugerechnet. Unmögliche Entscheidungen unterbrechen diese Ketten.

3.6. Simulationssteuerung und Berechnungszeit

Die Durchführung von Simulationsläufen wird in Plant Simulation gesteuert. Ausgangslage für einen neuen Lauf ist ein zurückgesetztes Modell mit Agenten, deren Steuerungslogik definiert ist. Das Socket Netzwerk muss bei der Verwendung der selbstlernenden Heuristik aktiviert sein.

Mittels der in Plant Simulation vorhandenen Standardklasse EventController können Einstellungen getroffen werden, über welchen Zeitraum das Produktionssystem simuliert werden soll. Die Klasse ExperimentManager kann verwendet werden, um mehrere

Simulationsläufe automatisiert aufeinanderfolgend durchzuführen. Hierbei können auch Einflussfaktoren von Lauf zu Lauf verändert werden.

Ein wichtiger Faktor für die Praktikabilität des Simulationsmodells ist die notwendige Berechnungszeit für die Durchführung von Experimenten. Je kürzer diese Berechnungszeit, umso schneller können Erkenntnisse gewonnen werden.

Die für die Simulationsläufe notwendige Berechnungszeit setzt sich aus den Berechnungszeiten in Plant Simulation und in Python zusammen. Das Simulationsmodell in Plant Simulation pausiert bis der Python-Algorithmus eine Entscheidung rückmeldet. In Tabelle 4 sind die Berechnungszeiten bei der Ausführung eines Agenten aufgeführt. Die Zeiten wurden bei deaktivierter Lernfähigkeit und Entscheidung durch die neuronalen Netze ermittelt.

	Wartezeit für Rückmeldung aus Python (ms)	Ausführungszeit der Methode DeepRL (ms)	Verhältnis
Mittelwert	17,23	20,98	82,50%
Standardabweichung	2,31	3,18	5,06%
Konfidenzbreite ($\alpha=5\%$)	0,46	0,63	0,54%
untere Grenze	16,78	20,35	81,96%
obere Grenze	17,69	21,61	83,04%

Tabelle 4: Berechnungszeiten bei Aktionswahl durch die neuronalen Netze (A35)

Es ist ersichtlich, dass im Mittel über 80% der Zeit auf eine Rückmeldung aus Python gewartet wird. Die Ausführung der Methoden innerhalb von Plant Simulation nimmt nur einen geringen Teil der gesamten Berechnungszeit in Anspruch. Beispielsweise benötigt die Methode mStep (eine Kernmethode des selbstlernenden Agenten) durchschnittlich nur 2,94ms je Ausführung eines Agenten (siehe Abbildung 13).

%Global	%Lokal	Zeit [s]	Aufrufe	ms/Aufruf	Methode
83.3%		8.094s	2752	2.94ms	.ReinforcementLearning.Objects.Agent_DeepRL.mStep
4.2%		0.412s	2752	0.15ms	.ReinforcementLearning.Objects.Agent_DeepRL.mWriteNewStates
4.2%		0.408s	2752	0.15ms	.ReinforcementLearning.Objects.Agent_DeepRL.mSocket_predict_action
2.7%		0.265s	2752	0.10ms	.ReinforcementLearning.Objects.Agent_DeepRL.mWriteStates
2.5%		0.243s	2752	0.09ms	.Models.Model.Socket_Server.callback
2.0%		0.194s	2752	0.07ms	.ReinforcementLearning.Objects.Agent_DeepRL.mStatistics

Abbildung 13: 6 Methoden mit dem größten CPU-Zeitverbrauch in Plant Simulation (A35)

4. Experimentieren mit dem Simulationsmodell

Um die Frage nach der Leistungsfähigkeit des selbstlernenden Multiagentensystems zu beantworten, wird mit dem Simulationsmodell experimentiert. Die durchgeführten Experimente führen zu neuen Erkenntnissen. Diese zeigen teilweise Einschränkungen des zugrundeliegenden Modells auf oder weisen auf interessante Einflussfaktoren hin. Für weiterführende Experimente bietet es sich demnach an, das Modell weiterzuentwickeln. Aus diesem Grund existieren zum Zeitpunkt dieses Berichts ca. 40 verschiedene Modellversionen.

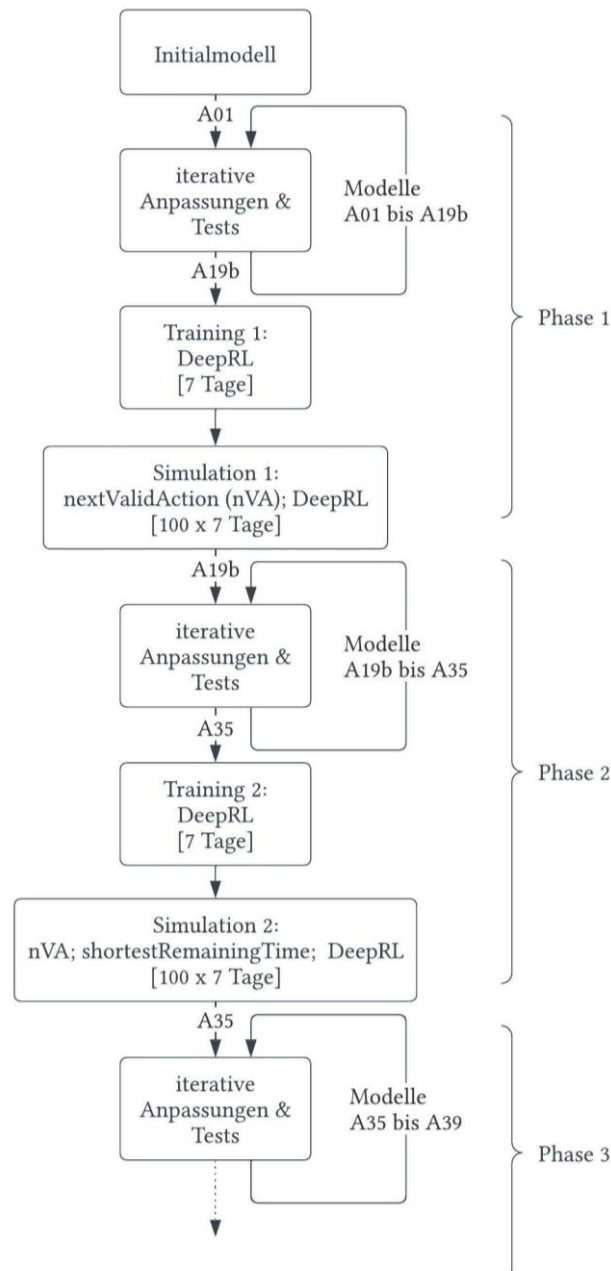


Abbildung 14: Ablauf der Experimente

Das für diese Arbeit gewählte Vorgehen wird in Abbildung 14 verständlich. Darin sind 3 Phasen zu erkennen. In diesen Phasen werden zu Beginn schrittweise neue Versionen des Simulationsmodells entwickelt. Neue Funktionalitäten und andere Einstellungen der

Einflussfaktoren werden durch Experimente mit geringem Aufwand (Tests) überprüft, bevor die nächste Modellversion aufgelegt wird. Nach dieser iterativen Weiterentwicklung werden mit der jeweils aktuellen Modellversion ausführliche Simulationsexperimente (Simulation 1 und 2) durchgeführt. Dabei werden 100 Simulationsläufe über 7 Tage Produktion je Steuerungsheuristik simuliert. Mittels dieser ausführlichen Simulationen können anschließend vergleichende Aussagen über die Leistungsfähigkeit der Heuristiken getroffen werden.

Die Lernfähigkeit der Heuristik DeepRL ist bei diesen Simulationsläufen abgeschaltet. Auf diese Weise wird die Vergleichbarkeit zwischen den verschiedenen Simulationsläufen erhalten. Die Heuristik wird zuvor über einen Simulationszeitraum von 7 Tagen Produktion trainiert (Training 1 und 2).

4.1. Erste Phase: Nachweis der Lernfähigkeit

In der ersten Phase wurde das Simulationsmodell entsprechend den in Kapitel 3 vorgestellten Grundsätzen entwickelt. Allerdings weichen die Modellversionen der Phase 1 in manchen Teilen vom beschriebenen aktuellen Zustand ab. Nachfolgend eine Übersicht der wichtigsten Abweichungen für das Modells A19b:

- Das Produktionssystem wurde mit nur drei Eingangspuffer-Maschinen Paaren modelliert (siehe hierzu auch Abbildung 15)
- Auf dem Förderband befinden sich nur 12 Ladungsträger.
- Zum Leistungsvergleich steht nur die Heuristik nextValidAction der selbstlernenden Strategie gegenüber.
- Die zufällige Verteilung des Zeitbedarfs für Bearbeitungsschritte liegt mit Sigma 15 min und einer Obergrenze von 60 min höher.

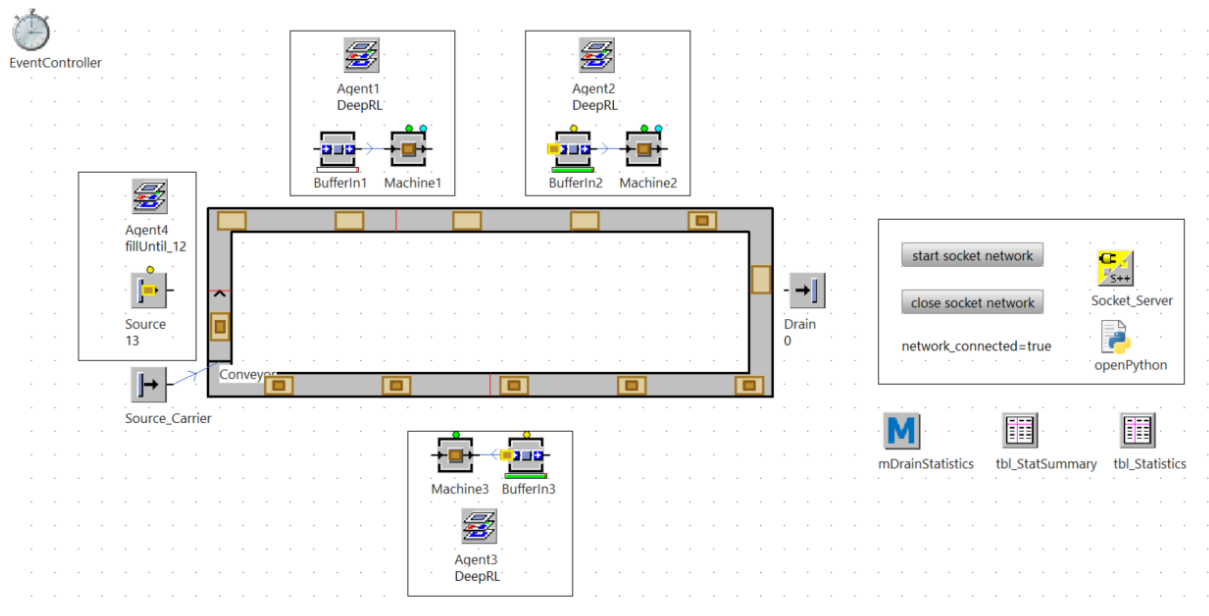


Abbildung 15: 2D-Darstellung des Simulationsmodells A19b

Während des Trainings (Training 1) kann für alle 3 Agenten die in Abbildung 16 dargestellte Entwicklung der Belohnung beobachtet werden. Der genaue Verlauf des gleitenden

Mittelwerts über die letzten 100 Entscheidungen ist zwar individuell, folgt aber im Wesentlichen für alle Agenten der gleichen Entwicklung.

Nach anfänglich deutlich negativen Werten lässt sich ab ca. 1000 Entscheidungen ein steiler Anstieg feststellen. Anschließend pendeln sich die Werte im positiven Bereich der y-Achse ein. Negative Werte sind selten. Die Werte streuen hauptsächlich zwischen 0 und +2.

Die anfänglich deutlich negativen Werte lassen sich dadurch erklären, dass zu Beginn des Trainings der Algorithmus nur zufällige Entscheidungen trifft, um möglichst explorativ vorzugehen. Auf diese Weise werden viele unmögliche Aktionen gewählt und entsprechend bestraft. Erst wenn im Datenspeicher über 1000 Status-Aktion-Folgestatus-Belohnung Kombinationen gespeichert sind, wird Epsilon schrittweise reduziert. Dadurch werden die Entscheidungen zunehmend durch das neuronale Netz getroffen und dieses wird trainiert.

Durch den Anstieg der mittleren Belohnung lässt sich schließen, dass der Algorithmus lernt mögliche Aktionen zu wählen. Bestrafungen (negative Werte) werden reduziert. Zusätzlich lässt sich durch die vorwiegend positiven Werte schließen, dass Transfers durchgeführt werden. Der Algorithmus lernt somit nicht nur, dass kein Transfer immer möglich ist (Belohnung 0). Mit dieser Beobachtung lässt sich somit nachweisen, dass durch Selbstlernen eine nutzbare Heuristik entstanden ist.



Abbildung 16: gleitender Durchschnitt an Belohnung über die letzten 100 Entscheidungen (Agent 1, A19b)

Um die Leistungsfähigkeit einer solchen selbstgelernten Heuristik zu überprüfen, dient der Vergleich mit der einfachen Heuristik nextValidAction. In Simulation 1 wurden hierzu jeweils 100 Simulationsläufe über 7 Tage Produktion durchgeführt. Das Ergebnis ist in Tabelle 5 zu sehen.

Der gepaarte t-Test ergibt für die Differenz der Wochenproduktionsmenge ein 95%-Konfidenzintervall von $[-2,3; -0,23]$. Entsprechend ist anzunehmen, dass die selbstgelernte Heuristik minimal schlechtere Leistung bringt als nextValidAction. De facto sind beiden Steuerungsansätze aber gleichwertig.

	nextValidAction	DeepRL	Differenz
Mittelwert	359,51	358,24	-1,27
Standardabweichung	9,13	8,99	5,23

Konfidenzbreite ($\alpha = 5\%$)	1,81	1,78	1,04
untere Grenze	357,70	356,46	-2,31
obere Grenze	361,32	360,02	-0,23

Tabelle 5: Leistungsvergleich Phase 1

Bei der Analyse der Systemressourcen zeigt sich, dass Maschine 3 der Engpass des Systems ist. Siehe hierzu Abbildung 17. Die Auslastung liegt bei ca. 95% und der „Blockiert-Anteil“ bei ca. 3%. Damit bleibt nicht mehr viel Spielraum für die Verbesserung des Durchsatzes durch andere Steuerungsansätze. Das System ist mit den eingesetzten Steuerungsheuristiken nahezu ausgereizt.

Zur weiteren Untersuchung der Leistungsfähigkeit von selbstlernenden Agenten bedarf es eines komplexeren Produktionssystems. Es sollte die Möglichkeit bestehen, dass der selbstlernende Algorithmus die einfache Heuristik nextValidAction übertrifft.

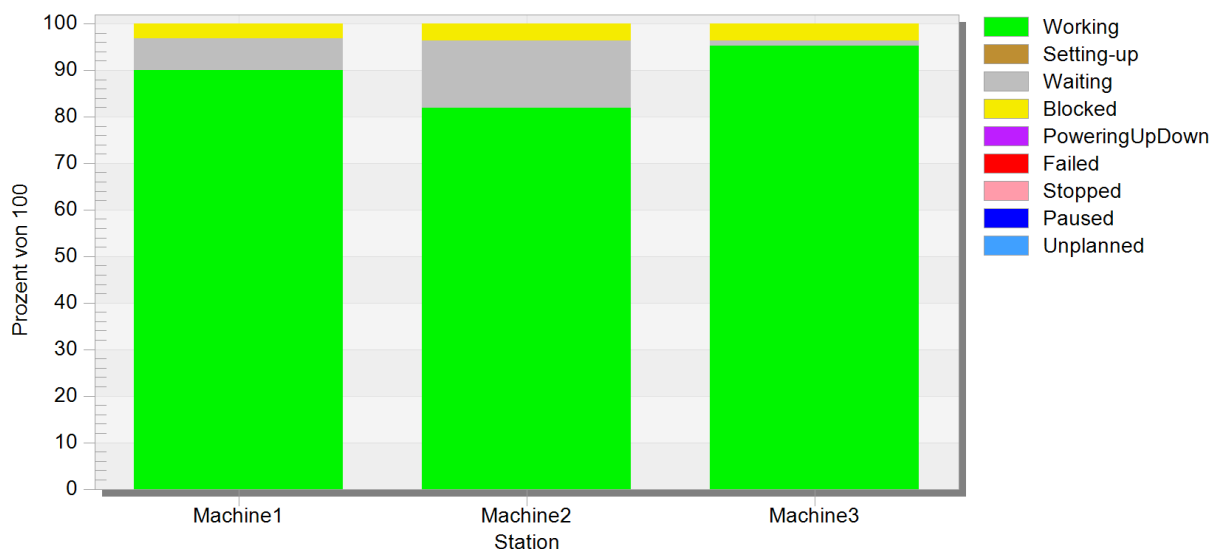


Abbildung 17: Auslastung der Maschinen unter Verwendung der Heuristik nextValidAction (A19b)

4.2. Zweite Phase: Komplexitätssteigerung

Entsprechend dem identifizierten Bedarf nach mehr Komplexität wurde das Modell in Phase 2 weiterentwickelt. Hierzu wurden diverse Parameter mittels statistisch nicht abgesicherter Tests untersucht und manche abgeändert. Hier eine Auswahl der relevantesten Parameter:

- Eingangspuffergröße
- Anzahl Aufträge im System
- Anzahl Agenten (Maschine-Puffer Paare)
- Zufallsverteilungen der Bearbeitungszeiten
- Erholzeit der Maschinen

Die resultierende Modellversion A35 entspricht dem in dieser Arbeit vorgestellten aktuellen Stand. Im Vergleich zu Phase 1 enthält dieses Modell zusätzlich die weitere Heuristik „shortestRemainingTime“ zum Leistungsvergleich.

Während des Trainings entsteht für die selbstlernenden Agenten in Phase 2 eine vergleichbare Entwicklung der gleitenden mittleren Belohnung, wie in Phase 1. Siehe hierzu

Abbildung 18. Die Lernfähigkeit ist somit auch in Phase 2 nachweisbar. Auffällig ist jedoch, dass innerhalb der 7 Tage Produktionszeit deutlich mehr Entscheidungen durch den Agenten getroffen werden. Statt der vormals ca. 24000 Aufrufe wird der Agent in Phase 2 ca. 80000-mal befragt. Die Lernkurve ab ca. 1000 Entscheidungen ist demnach im Diagramm deutlich gestaucht. Diese Vermehrung der Aufrufe ist auf die Zunahme der zirkulierenden Ladungsträger zurückzuführen. Immer wenn ein solcher den Sensor des Agenten passiert, ist eine Entscheidung durch den Agenten zu treffen.

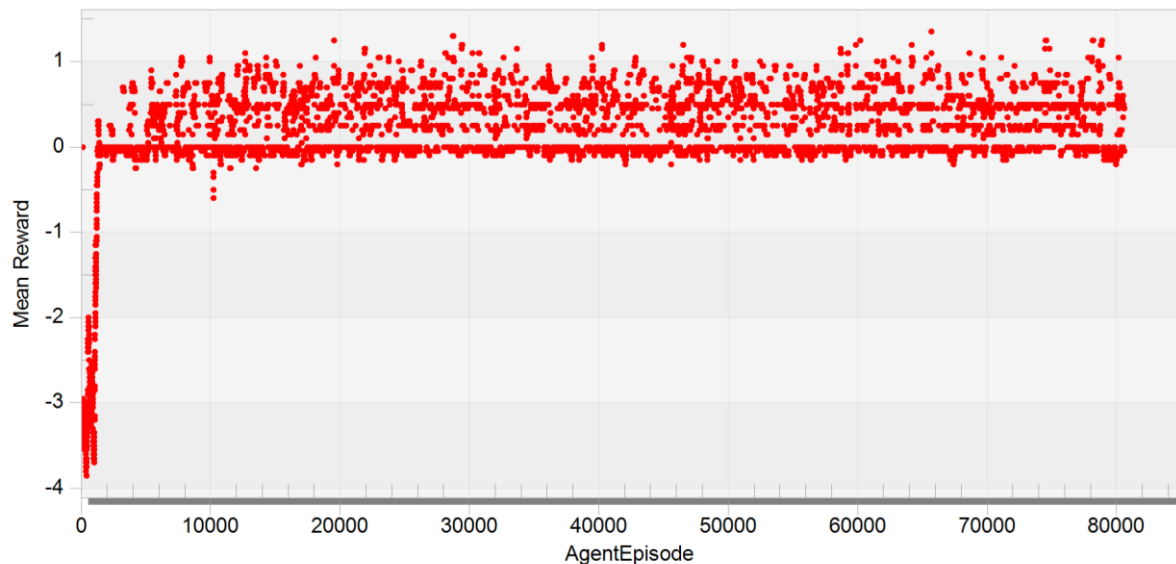


Abbildung 18: gleitender Durchschnitt an Belohnung über die letzten 100 Entscheidungen (Agent 1, A35)

In Abbildung 19 ist im Vergleich zu Abbildung 17 eine Reduktion der Engpassauslastung zu erkennen. Auch die leistungstärkste Heuristik (shortestRemainingTime) reizt in Simulation 2 das System nicht bis an die Grenzen aus. Die Komplexitätssteigerung ist demnach erfolgreich umgesetzt worden.

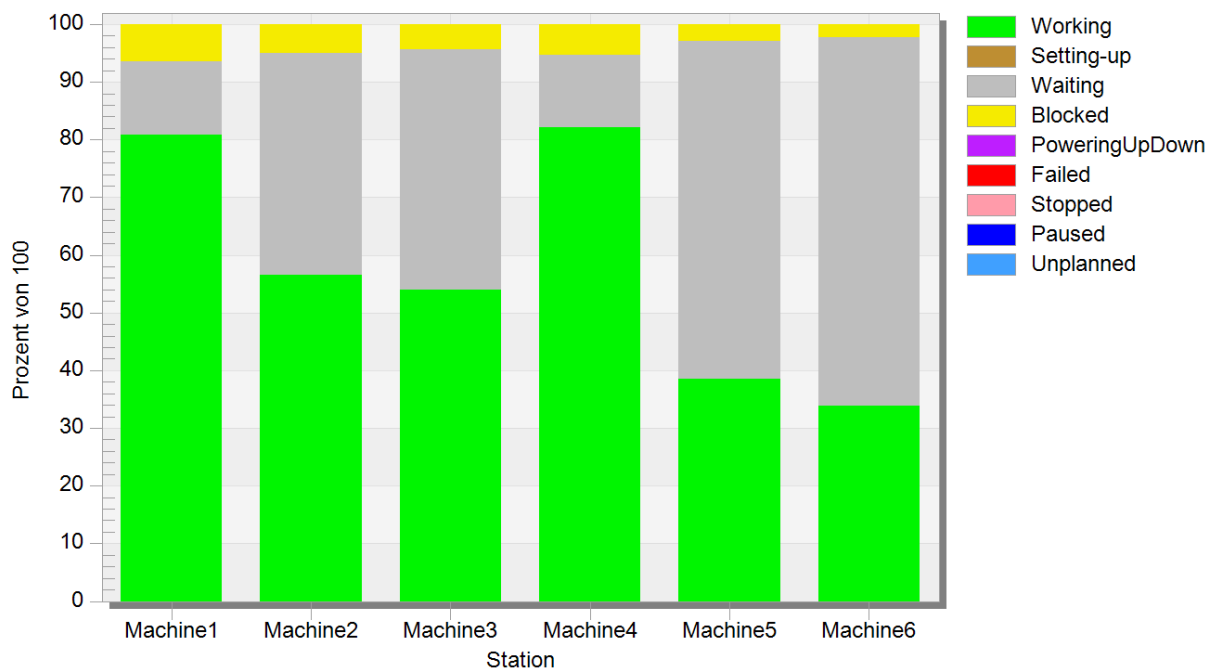


Abbildung 19: Auslastung der Maschinen unter Verwendung der Heuristik shortestRemainingTime (A35)

In Tabelle 6 ist ein Vergleich der Leistung der drei verfügbaren Steuerungsansätze verdeutlicht. Die Heuristiken DeepRL und nextValidAction sind erneut nahezu gleichwertig. Die Heuristik shortestRemainingTime weist mit dem 95%-Konfidenzintervall von [673,84; 678,48] eine signifikant höhere Leistung auf.

Demnach ist die Interpretation zulässig, dass es dem selbstlernenden Algorithmus bisher nur gelingt, ähnlich nextValidAction, die unmöglichen Entscheidungen zu vermeiden und die möglichen Transfers durchzuführen. Eine Zusammenarbeit der Agenten untereinander und eine sinnvolle Priorisierung der Aufträge wird bisher nicht gelernt.

	DeepRL	nextValidAction	shortestRemaining Time
Mittelwert	637,79	638,30	676,16
Standardabweichung	12,02	12,04	11,72
Konfidenzbreite ($\alpha = 5\%$)	2,38	2,39	2,32
untere Grenze	635,41	635,91	673,84
obere Grenze	640,17	640,69	678,48

Tabelle 6: Leistungsvergleich Phase 2

Diese Ergebnisse zeigen, dass bereits mit einem einfachen selbstlernenden Multiagentensystemen eine primitive Steuerungslogik zur Produktionssteuerung gelernt werden kann. Diese trifft mögliche, jedoch nicht optimierte Entscheidungen.

Das Modell weist durch die gesteigerte Komplexität Optimierungspotential für die Steuerungsansätze auf. Um die Fähigkeit des selbstlernenden Ansatzes zu mehr Leistung zu überprüfen, gilt es, den Algorithmus zu verbessern.

4.3. Dritte Phase: Möglichkeiten zur Leistungssteigerung

Ziel der dritten Phase ist die Leistungssteigerung der selbstgelernten Steuerungsansätze. Die Lernfähigkeit der Agenten hängt in Summe von einer Vielzahl von Faktoren ab. Im aktuellen Zustand führt der resultierende Steuerungsansatz alle möglichen Transfers direkt durch. Dieser Zustand ist ein lokales Optimum, über das hinaus keine bessere Strategie gelernt wird.

Ziel ist es, die Einflussfaktoren so zu verändern, dass bessere Heuristiken zur Produktionssteuerung gelernt werden. Wie in den beiden vorangegangenen Phasen bietet sich ein iteratives, exploratives Verfahren zur Erkundung der Einflüsse an. In einem ersten Schritt wurden die folgenden Einflussfaktoren als interessant identifiziert:

- Belohnungsfunktion
- Design der neuronalen Netze
- Epsilon-Greedy-Policy
- Lernrate
- Skontorate (Gamma)

Das Modell wurde zur Untersuchung dieser Einflussfaktoren entsprechend weiterentwickelt. Die identifizierten Faktoren können in den neuen Modellversionen mit geringem Aufwand verändert werden, um Versuche mit unterschiedlichen Kombinationen durchzuführen.

Eine zentrale Rolle spielt für das Lernverfahren die *Belohnungsfunktion*. Aktuell wird eine Belohnungsfunktion eingesetzt, die lediglich die lokale Statusveränderung zur Grundlage für

die Belohnung verwendet. Nur Veränderungen, die direkt durch den Agenten beeinflusst werden können, werden berücksichtigt. Im Vergleich hierzu enthalten die Belohnungsfunktionen der vorgestellten Beispiele aus der Literatur auch einen globalen Anteil. Hierbei geht die direkte Ursache-Wirkungs-Beziehung verloren und das Gesamtverhalten des Systems gewinnt an Wichtigkeit. Entsprechende Anpassungsvorschläge sind in Tabelle 7 zu finden. Diese legen die zu optimierende Kenngröße Durchsatz direkt für die Ermittlung der Belohnung zugrunde.

Ein ergänzender Vorschlag zur Anpassung der Belohnungsfunktion ist die Bestrafung (geringe negative Belohnung) der Entscheidung „kein Transfer“. Damit soll verhindert werden, dass die Agenten lernen, dass „kein Transfer“ immer möglich ist und in diesem lokalen Optimum verharren.

Ein weiterer Einflussfaktor ist die Komplexität der eingesetzten *neuronalen Netze*. Eventuell können komplexere Netze zu komplexeren Steuerungslogiken führen. Verändert werden kann die Anzahl der Zwischenschichten (hidden layer). Außerdem kann die Anzahl der Neuronen je Schicht angepasst werden. In Tabelle 7 werden drei Alternativen für die Gestaltung der neuronalen Netze aufgeführt. Alle Vorschläge weisen mehr Neuronen auf als die aktuelle Ausführung und sind somit komplexer.

Die *Epsilon-Greedy-Policy* entscheidet darüber wie viel Exploration durch zufällige Entscheidungen stattfindet. Prinzipiell sinkt die Wahrscheinlichkeit für eine zufällige Entscheidung über die Trainingszeit. Die Strategie der Reduktion und die Mindestwahrscheinlichkeit kann verändert werden. Eine höhere Explorationsrate kann helfen, das lokale Optimum zu überwinden. (siehe Tabelle 7)

Die *Lernrate* beschreibt die Größe der Anpassungsschritte der neuronalen Netze bei der Aktualisierung. Größere Schritte, wie in Tabelle 7 vorgeschlagen, könnten helfen eine bessere Strategie zu finden. Die *Skontorate* dient der anteiligen Berücksichtigung von zukünftigen Belohnungen. Eine Erhöhung, wie in Tabelle 7 vorgeschlagen, erhöht die Weitsichtigkeit der Agenten. Somit werden langfristige Folgen der Entscheidungen mehr berücksichtigt.

Faktoren	Gamma	Lernrate	KNN-Design	Epsilon Greedy ($\epsilon/\text{decay}/\epsilon\text{-min}$)	Belohnungsfunktion
Mögliche Faktorstufen	0,9	0,001	10x24x24x3	1/0,99/0,01	Bestrafung (-5) für unmögliche Wahl; Belohnung (+25) für möglichen Transfer und ($10 \cdot x$) Anzahl an hintereinander durchführbaren Operationen, kein Transfer neutral (0)
	0,99	0,01	10x64x24x3	1/0,99/0,05	Bestrafung (-5) für unmögliche Wahl; Belohnung (+5) für möglichen Transfer; kein Transfer neutral; ($+100 \cdot x$) für x =Zuwachs des Durchsatzes seit letzter Entscheidung
	0,999	0,1	10x24x12x6x3	1/0,999/0,1	Bestrafung (-5) für unmögliche Wahl; kein Transfer Bestrafung (-2); ($+100 \cdot x$) für x =Zuwachs des Durchsatzes seit letzter Entscheidung
			10x48x24x12x3		Bestrafung (-5) für unmögliche Wahl; Belohnung ($+100 \cdot x$) für x =Zuwachs des Durchsatzes seit letzter Entscheidung

Tabelle 7: Einflussfaktoren und Faktorstufen

Die erste Zeile von Tabelle 7 bildet die aktuellen Einstellungen der fünf Faktoren ab. Die folgenden Zeilen listen mögliche Faktorstufen für Versuche mit dem Lernalgorithmus auf.

Es bietet sich an, die vorgestellten Einflussfaktoren und deren Wechselwirkungen mittels Versuchsplanung systematisch zu untersuchen. Diese Untersuchungen konnten im Rahmen der vorliegenden Arbeit nicht mehr durchgeführt werden.

Ergebnisse, die auf gesteigerte Leistung der selbstgelernten Steuerungsansätze schließen lassen, sind wie in Phase 1 und 2 statistisch abzusichern.

4.4. Kritik am Simulationsmodell

Für die Nutzbarkeit des Simulationsmodells zu weiteren Experimenten ist die hohe Berechnungszeit für Simulationsläufe mit selbstlernender Heuristik störend. Als zeitintensiv konnte in Kapitel 3.6 die Ausführung der Python-Skripte identifiziert werden. Dieses Problem könnte durch eine schnellere Schnittstelle eventuell verringert werden. Alternativ könnte die Lernumgebung direkt in Plant Simulation implementiert werden. Auf diese Weise könnte auf eine Schnittstelle vollständig verzichtet werden. Ob dies möglich ist, muss jedoch überprüft werden. Eine weitere Möglichkeit wäre der Einsatz von weniger Werkstückträgern innerhalb des Produktionssystems. Diese Maßnahme würde die notwendige Anzahl an Entscheidung durch den Agenten reduzieren.

In der bisherigen Ausführung sorgen Entscheidungen für unmögliche Aktionen zum Abbruch von Entscheidungsketten, anhand derer gelernt werden kann. Durch die Korrektur mittels zulässiger Aktion existiert im Simulationsmodell jedoch eine Entscheidungskette, die zum Lernen eines weiteren Zeithorizonts genutzt werden könnte. Diese längerfristige Kette wird bisher nicht für das Lernverfahren genutzt. Insbesondere zu Beginn des Trainingslaufs werden viele unzulässige Entscheidungen getroffen. Die längerfristige Entscheidungskette könnte hier den Lernprozess bezüglich Optimierung auf langfristigen Erfolg unterstützen.

Die Experimente der Arbeit beschränken sich bisher auf gleichbleibende Produktionssysteme mit gleichbleibender Problemstellung. Als Vorteile agentifizierter Systeme werden jedoch häufig deren Flexibilität und Reaktionsfähigkeit bei Veränderungen hervorgehoben. Die Integration solcher Veränderungen in die Experimente könnte lohnend sein. Insbesondere bei aktivierter Lernfähigkeit wäre es aufschlussreich, zu beobachten, wie der Steuerungsansatz sich bei Veränderungen anpasst. Im Vergleich zu statischen Heuristiken könnten hier Leistungsvorteile entstehen.

Leistung wird in dieser Arbeit, wie eingangs definiert, nur mittels des Durchsatzes bestimmt. In der Praxis ist es jedoch notwendig, neben dem Durchsatz weitere Leistungskriterien zu beachten. Bei weiterführenden Experimenten wäre es für gesamtheitlichere Leistungsvergleiche notwendig, zusätzliche Kriterien einzuführen. Beispielsweise könnte parallel zum Durchsatz auf Kriterien wie Maschinenauslastung oder Termineinhaltung optimiert werden.

5. Zusammenfassung und Ausblick

Ziel dieser Arbeit war es, anhand eines virtuellen Simulationsmodells herauszufinden, wie leistungsfähig ein selbstlernendes Multiagentensystem in der Produktionssteuerung sein kann.

Hierzu wurde der selbstlernende Algorithmus mit statischen Steuerungsheuristiken verglichen. Leistungskriterium war der Durchsatz einer 7-Tage-Produktion. Die grundlegende Struktur der selbstlernenden Modellbestandteile wurde von einer Vorgängerarbeit im Bereich der Einzelagentensysteme übernommen und angepasst. Das Modell wurde im Laufe der durchgeführten Experimente weiterentwickelt, um Einschränkungen zu überkommen.

In den durchgeführten Experimenten zeigte sich, dass der selbstlernende Algorithmus das Leistungsniveau eines einfachen Steuerungsansatzes erreicht. Mögliche Bearbeitungsschritte an den vorliegenden Aufträgen wurden im fiktiven Produktionssystem durchgeführt. Eine Leistungssteigerung durch erfolgreiche Priorisierung der Aufträge und Kollaboration zwischen den einzelnen Agenten wurde mit den Ausgangseinstellungen nicht erreicht.

Für potenzielle Leistungsverbesserung ist das Lernverfahren anzupassen. Die Einflussfaktoren auf das Lernverfahren sind vielfältig. Entsprechende Faktoren und sinnvolle Faktorstufen wurden identifiziert. Insbesondere Änderungen an der Belohnungsfunktion könnten hier vielversprechend sein. Beispiele aus der Literatur legen nahe, dass zumindest ein Teil der Belohnung direkt durch globale Systemveränderungen begründet sein sollte.

Eine weitergehende experimentelle Untersuchung der Einflussfaktoren konnte im Rahmen der Arbeit nicht mehr durchgeführt werden. Der Nachweis, dass das selbstlernende Multiagentensystem die Leistung eines einfachen Steuerungsansatzes übersteigen kann, ist demnach noch zu erbringen.

Zu beachten ist, dass die vorliegende Arbeit beim Leistungsvergleich auf den Durchsatz eingeschränkt ist. In der Praxis sind neben dem Durchsatz auch weitere Leistungskriterien wie beispielsweise Maschinenauslastung oder Termintreue zu berücksichtigen. Weitergehend sind in den bisherigen Experimenten potenzielle Vorteile von selbstlernenden agentifizierten Systemen nicht überprüft worden. Insbesondere die Anpassungsfähigkeit bei sich ändernden Bedingungen wird häufig als Vorteil solcher Systeme beschrieben und ist nicht berücksichtigt.

In Bezug auf die Forschungsfrage lässt sich feststellen, dass die Implementierung eines selbstlernenden Multiagentensystems zur Produktionssteuerung mit bestehenden Softwarebausteinen in einem Simulationsmodell einfach möglich ist. Das Lernen eines hochleistungsfähigen Steuerungsansatzes bedarf jedoch vermutlich eines detailliert auf den Anwendungsfall abgestimmten Lernverfahrens. Nach welchen Regeln ein Lernverfahren aufzubauen ist, damit höhere Leistung erreicht wird, gilt es in weiteren Arbeiten zu herauszufinden.

Literaturverzeichnis

- Bauernhansl, Thomas (2014): Komplexe Märkte erfordern komplexe Fabrik- und Managementstrukturen. Vielfalt ist Trumpf, aber nur wenn man mit ihr umgehen kann. In: interaktiv. Fraunhofer - Institut für Produktionstechnik und Automatisierung (IPA) (1), S. 36–39. Online verfügbar unter <https://www.ipa.fraunhofer.de/de/Kompetenzen/unternehmensstrategie-und-entwicklung/lean-management/komplexitaetsbewirtschaftung.html>, zuletzt geprüft am 01.10.2021.
- Chen, Ci; Xia, Beixin; Zhou, Bing-hai; Xi, Lifeng (2015): A reinforcement learning based approach for a multiple-load carrier scheduling problem. In: *J Intell Manuf* 26 (6), S. 1233–1245. DOI: 10.1007/s10845-013-0852-9.
- Claus, Thorsten; Herrmann, Frank; Manitz, Michael (Hg.) (2021): Produktionsplanung und -steuerung. Forschungsansätze, Methoden und Anwendungen. Springer-Verlag GmbH. 2., überarbeitete und erweiterte Auflage. Berlin, Heidelberg: Springer Gabler. Online verfügbar unter <http://www.springer.com/>.
- Crites, Robert H.; Barto, Andrew G. (1998): Elevator Group Control Using Multiple Reinforcement Learning Agents. In: *Machine Learning* 33 (2/3), S. 235–262. DOI: 10.1023/A:1007518724497.
- Dudenredaktion (o. J.): selbstlernend. Hg. v. Duden online. Online verfügbar unter <https://www.duden.de/rechtschreibung/selbstlernend>, zuletzt geprüft am 29.12.2021.
- Frochte, Jörg (2019): Maschinelles Lernen. Grundlagen und Algorithmen in Python. 2., aktualisierte Auflage. München: Hanser.
- Gutenschwager, Kai; Rabe, Markus; Spieckermann, Sven; Wenzel, Sigrid (2017): Simulation in Produktion und Logistik. Grundlagen und Anwendungen. Berlin, Heidelberg: Springer Vieweg.
- Hajduk, Mikuláš (2019): Cognitive Multi-Agent Systems. Structures, Strategies and Applications to Mobile Robotics and Robosoccer. Unter Mitarbeit von Marek Sukop und Matthias Haun. Cham: Springer International Publishing AG (Studies in Systems, Decision and Control Ser, v.138).
- Hofmann, Constantin; Krahe, Carmen; Stricker, Nicole; Lanza, Gisela (2020): Autonomous production control for matrix production based on deep Q-learning. In: *Procedia CIRP* 88, S. 25–30. DOI: 10.1016/j.procir.2020.05.005.
- Koren, Yoram (2010): The Global Manufacturing Revolution. Hoboken, NJ, USA: John Wiley & Sons, Inc.
- Kuhnle, Andreas; Kaiser, Jan-Philipp; Theiß, Felix; Stricker, Nicole; Lanza, Gisela (2021): Designing an adaptive production control system using reinforcement learning. In: *J Intell Manuf* 32 (3), S. 855–876. DOI: 10.1007/s10845-020-01612-y.
- Martin Schlump (2020): Simulationsbasiertes maschinelles Lernen für die agentenbasierte Steuerung von Materialflüssen. Konzeption und prototypische Umsetzung. Masterarbeit. Technische Hochschule Ulm.

Meudt, Tobias; Wonnemann, Andreas; Metternich, Joachim (2017): Produktionsplanung und -steuerung (PPS) - ein Überblick der Literatur der unterschiedlichen Einteilung von PPS-Konzepten. Technische Universität Darmstadt. Online verfügbar unter https://tuprints.ulb.tu-darmstadt.de/6654/13/20170904_Literatur%C3%BCberblick%20zur%20Einteilung%20von%20PPS-Konzepten_v2.pdf, zuletzt geprüft am 27.12.2021.

Otte, Ralf (2019): Künstliche Intelligenz für Dummies. 1. Auflage. Weinheim: Wiley-VCH Verlag GmbH & Co. KGaA (...für Dummies).

Scholz-Reiter, Bernd; Höhns, Hartmut (2006): Selbststeuerung logistischer Prozesse mit Agentensystemen. In: Günther Schuh (Hg.): Produktionsplanung und -steuerung. Grundlagen, Gestaltung und Konzepte. 3., völlig neu bearb. Aufl. Berlin: Springer (VDI-Buch), S. 745–780.

Siepermann, Christoph (2018): Produktionsplanung und -steuerung. Definition: Was ist "Produktionsplanung und -steuerung"? Hg. v. Gabler Wirtschaftslexikon. Online verfügbar unter <https://wirtschaftslexikon.gabler.de/definition/produktionsplanung-und-steuerung-51585/version-274746>, zuletzt aktualisiert am 15.02.2018, zuletzt geprüft am 29.12.2021.

VDI-Richtlinien 3633, 2014: Simulation von Logistik-, Materialflussund Produktionssystemen.

Zipfel, Elias Josua (2022): masterProject. Production control using self-learning multi-agent-systems. Online verfügbar unter <https://github.com/ZipfelE/masterProject>, zuletzt geprüft am 18.03.2022.