h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbi
FACHBEREICH INFORMATIK

h_da
HOCHSCHULE DARMSTADT
UNIVERSITY OF APPLIED SCIENCES
fbmn
FACHBEREICH MATHEMATIK
UND NATURWISSENSCHAFTEN

Hochschule Darmstadt

Fachbereiche Mathematik und Naturwissenschaften & Informatik

Dynamic Scheduling of Gantry Robots using Cooperative Multi-Agent Reinforcement Learning

Abschlussarbeit zur Erlangung des akademischen Grades

Master of Science (M. Sc.)

im Studiengang Data Science

vorgelegt von

Jannik Felix Hinrichs

Matrikelnummer: 1114333

Referent	:	Prof. Dr. Horst Zisgen
Korreferent	:	Prof. Dr. Arnim Malcherek

Ausgabedatum	:	11.03.2024
Abgabedatum	:	05.09.2024

DECLARATION

Ich versichere hiermit, dass ich die vorliegende Arbeit selbstständig verfasst und keine anderen als die im Literaturverzeichnis angegebenen Quellen benutzt habe.

Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder noch nicht veröffentlichten Quellen entnommen sind, sind als solche kenntlich gemacht.

Die Zeichnungen oder Abbildungen in dieser Arbeit sind von mir selbst erstellt worden oder mit einem entsprechenden Quellennachweis versehen.

Diese Arbeit ist in gleicher oder ähnlicher Form noch bei keiner anderen Prüfungsbehörde eingereicht worden.

Darmstadt, September 5, 2024

Jannik Felix Hinrichs

ABSTRACT

In recent years, there has been a notable increase in the attention given to Machine Learning and Reinforcement Learning in both research and practice. In the manufacturing industry, for instance, an increasing number of approaches are being investigated with the objective of determining how artificial intelligence can be utilized to enhance efficiency and productivity. One potential area of application is the flexible control and rapid adaptation of production systems to changing conditions through Reinforcement Learning. In contrast to conventional, purely rule-based control systems, this is only possible with great effort. Digital twins are now being used in the development of production systems, allowing risks during subsequent operations to be identified and minimized at an early stage. The simulation software used for this is particularly suitable as an environment for Reinforcement Learning approaches.

Previous research has demonstrated that Reinforcement Learning with a single agent can be utilized to control gantry robots in relatively straightforward production lines. However, in more advanced production lines with multiple gantry robots, the exponentially expanding state space presents a significant challenge for the Single-Agent approach, ultimately reducing its efficiency. Furthermore, the asynchronous execution of actions with multiple agents introduces additional complexities for the Reinforcement Learning approach that require further investigation and resolution.

This thesis presents a Multi-Agent Reinforcement Learning approach in which each gantry robot is controlled by one agent. An analysis of previous research forms the basis for identifying the major challenges. Based on this, the methodology of the Multi-Agent approach is developed and explained. The effectiveness of the presented methodology is then tested and evaluated in several iterative experiments.

The experiments demonstrate the efficacy of the methodology in controlling gantry robots in a production plant. Despite training the agents independently, they learn a strategy for cooperative collaboration. The use of local state and action spaces for the respective agents has a negligible impact on the learned strategy. Additionally, it is shown that adapting the reward function of the Reinforcement Learning approach effectively compensates for inaccuracies in the simulation software.

keywords: Reinforcement Learning, Cooperative Multi-Agents, Gantry Robots, Dynamic Scheduling, Production Control

ZUSAMMENFASSUNG

In den vergangenen Jahren gewannen Machine Learning und Reinforcement Learning mehr Aufmerksamkeit in der Forschung und auch in der Praxis. So werden beispielsweise in der produzierenden Industrie vermehrt Ansätze untersucht, wie künstliche Intelligenz eingesetzt werden kann, um die Effizienz und die Produktivität zu steigern. Ein potenzielles Anwendungsgebiet ist die flexible Steuerung und schnelle Adaption von Produktionsanlagen an veränderte Bedingungen durch Reinforcement Learning. Mittels herkömmlicher rein regelbasierten Steuerungen gelingt dies nur mit großem Aufwand. Inzwischen werden bei der Entwicklung von Produktionsanlagen digitale Zwillinge eingesetzt, wodurch Risiken beim späteren Betrieb vorzeitig erkannt und minimiert werden können. Die dafür verwendete Simulationssoftware eignet sich in besonderem Maße als Environment für Reinforcement Learning Ansätze.

In vorangegangenen Forschungsarbeiten konnte nachgewiesen werden, dass Reinforcement Learning mit einem einzelnen Agenten zur Steuerung von Portalrobotern in simplen Produktionsanlagen anwendbar ist. Bei komplexeren Anlagen mit mehreren Portalrobotern führt der exponentiell wachsende Zustandsraum allerdings zu Herausforderungen für den Single-Agenten Ansatz, was die Effizienz des Ansatzes reduziert. Zudem ergeben sich durch die asynchrone Ausführung der Aktionen bei mehreren Agenten weitere Herausforderungen für den Reinforcement Learning Ansatz, die es zu bewältigen gilt.

Die vorliegende Masterarbeit präsentiert einen Multi-Agenten Reinforcement Learning Ansatz, bei dem jeder Portalroboter von einem Agenten kontrolliert wird. Die Analyse vergangener Forschungsarbeiten bildet die Grundlage für die Identifizierung der wesentlichen Herausforderungen. Davon ausgehend wird die Methodik des Multi-Agenten Ansatzes entwickelt und erläutert. Die Wirksamkeit der vorgestellten Methodik wird in mehreren iterativen Experimente getestet und bewertet.

Die durchgeführten Experimente belegen, dass die entwickelte Methodik sich erfolgreich zur Steuerung von Portalrobotern in einer Produktionsanlage eignet. Die Agenten erlernen, obwohl sie unabhängig voneinander trainiert werden, eine Strategie zur kooperativen Zusammenarbeit. Es konnte nachgewiesen werden, dass das Verwenden von lokalen Zustands- und Aktionsräumen für die jeweiligen Agenten nur marginale Auswirkungen auf die gelernte Strategie hat. Des Weiteren wurde demonstriert, dass durch Anpassen der Reward-Funktion des Reinforcement Learning Ansatzes Ungenauigkeiten in der Simulationssoftware effektiv kompensiert werden können.

Schlagwörter: Reinforcement Learning, Kooperative Multi-Agenten, Portalroboter, Dynamisches Scheduling, Produktionssteuerung

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LISTINGS

LIST OF ALGORITHMS

ACRONYMS

AI	Artificial Intelligence
ANN	Artificial Neural Network
DNN	Deep Neural Network
DQN	Deep Q-Network
FIFO	First In, First Out
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KISPo	KI-Verfahren zur Steuerung von Digitalen Portalroboterzwillingen
MA	Multi-Agent
MARL	Multi-Agent Reinforcement Learning
MTBF	Mean Time Between Failures
MTTR	Mean Time to Repair
MDP	Markov Decision Process
NN	Neural Network
SA	Single-Agent
SARL	Single-Agent Reinforcement Learning
TD	Temporal-Difference
RL	Reinforcement Learning

Part I

THESIS

1 INTRODUCTION

In the contemporary era, a considerable number of modern production facilities have adopted the use of robots to automate a range of tasks to enhance efficiency and productivity. For instance, these tasks encompass the distribution of resources within production facilities. In order to remain competitive in a dynamic global market, production facilities must be able to adapt quickly to new requirements. Traditionally, the transportation of goods was conducted through the use of rudimentary methods, such as First In, First Out (FIFO), or heuristic approaches. These straightforward methods are insufficient for maintaining the required adaptability, which is why dynamic scheduling is becoming increasingly popular. Dynamic scheduling is a methodology that employs, among other things, real-time decision-making based on the current condition of its environment.

In recent years, there has been a notable increase in research exploring the potential of Artificial Intelligence (AI) in optimizing dynamic scheduling problems. The simulation of actual production facilities with so-called digital twins – a technology employed under the rubric of Industry 4.0 – offers an optimal foundational basis for Reinforcement Learning (RL). The problem of distributing resources as a continuous task presents an additional challenge for RL, which is initially designed for problems with a fixed end.

The objective of the project *Dynamic Scheduling of Gantry Robots using Simulation and Reinforcement Learning* (german: *KI-Verfahren zur Steuerung von Digitalen Portalroboterzwillingen*) is to examine the potential of using RL to control gantry robots in a production line [Zis+24]. The production line represents a machining process comprising multiple work steps, such as drilling or milling. The gantry robots are responsible for the transportation of the workpieces between the processing machines. The objective of the RL approach is to learn a strategy that controls the actions of the gantry robots so that the achieved throughput is maximized. The initial work carried out as part of the project successfully demonstrated that an RL agent can learn such a strategy that is at least equivalent to simple conventional control concepts. This study has used a RL approach where one Single-Agent (SA) is responsible for the entire production system.

Nevertheless, challenges emerge when the simulated production system becomes more advanced, specifically when an increased number of work stations and portal robots are employed. One such challenge is that the individual portal robots are capable of performing their designated tasks in an asynchronous manner. However, communication between the simulation and the RL agent must be synchronized, as the agent is only capable of making a single decision at the same time. Furthermore, the strategy that must be learned is considerably more complex. On one hand, the environment's complexity rises in proportion to the number of components comprising the production line. On the other hand, the agent must learn a unified strategy to control all gantry robots, which are performing disparate actions concurrently. Moreover, the learned strat-

egy must learn both cooperation between the gantry robots and mutual consideration, as the portal robots operate in the same workspace. The functionality of the software utilized to simulate the digital twin presents a further challenge to the Single-Agent Reinforcement Learning (SARL) approach, as the transmitted environment states can be problematic. When multiple gantry robots are employed in the production line, this issue arises, which may negatively influence the RL algorithm. The challenges that arise are presented in detail in a subsequent chapter (see Chapter 5).

In accordance with the aforementioned challenges, this master's thesis will investigate whether these challenges can be overcome with the application of a Multi-Agent Reinforcement Learning (MARL) approach. It is essential to guarantee that the various agents learn to collaborate effectively in order to maximize the throughput of the produced workpieces through the implementation of a cooperative strategy. This strategy must also be evaluated in a dynamic environment, specifically in the context of random machine failures. Moreover, it is also necessary to ascertain whether the particular characteristic of the simulation software can be mitigated through the utilization of a Multi-Agent (MA) approach. The efficacy and robustness of the presented methodology will be evaluated through a series of experiments.

1.1 STRUCTURE OF THESIS

The following chapters of this thesis are structured as follows:

- Chapter 2 offers an overview of the core structural elements and operational characteristics of the simulated production plant, providing a foundation for subsequent analysis.
- Chapter 3 presents an examination of the diverse MARL approaches, elucidating their distinctive attributes and previous studies that have examined these approaches in different contexts.
- Chapter 4 explains the theoretical foundations of Neural Networks (NNs) and the concept of RL, which both are necessary to comprehend the proposed methodology.
- Chapter 5 provides an in-depth analysis of the specific challenges associated with the existing SARL approach. It then introduces the methodology of the proposed MARL approach, which has been designed to address these challenges.
- Chapter 6 outlines the key aspects of the implementation of the proposed MARL approach, offering a technical overview of how the methodology was realized in practice.
- Chapter 7 presents the conducted experiments and their subsequent evaluation. An analysis of the resulting data is provided in order to assess the performance and effectiveness of the proposed approach.
- Chapter 8 summarizes the findings of the thesis, reflecting on the insights gained from the research, and provides an outlook on potential future works.

2

PROJECT PRESENTATION

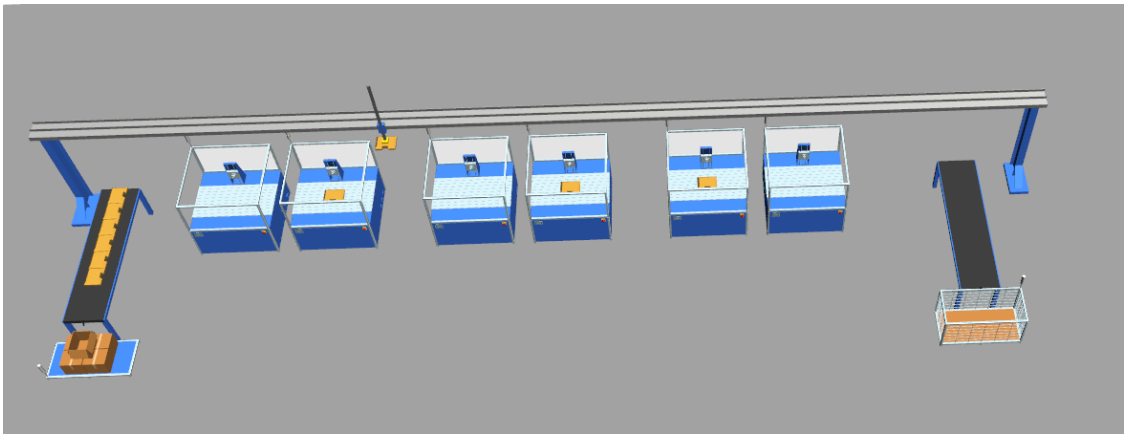

Figure 2.1: A digital twin of a production line with one single I-Loader and three work centers, each of which employs two work stations. The work stations are positioned between the input (left) and output (right) conveyor.

This thesis builds upon the fundamental structure of the project *Dynamic Scheduling of Gantry Robots using Simulation and Reinforcement Learning* (german: *KI-Verfahren zur Steuerung von Digitalen Portalroboterzwillingen (KISP0)*), which investigates the use of Reinforcement Learning (RL) for finding an optimal control of gantry robots for transporting workpieces in a production line [Mil+23]. The creation of a digital twin of production lines is facilitated by the utilization of simulation software, such as Plant Simulation from Siemens or AnyLogic. The digital twin of a production line, illustrated in Figure 2.1, comprises the following components:

- Gantry robots, also referred to as loaders in this thesis, are mobile robotic systems that navigate horizontally on a track situated above the machines. Two distinct types of loaders have been classified: the I-loader, which is equipped with one single gripper, and the H-loader, which is equipped with two independent grippers. The grippers are capable of vertical movement towards the machines and grabbing the workpieces. This thesis focuses exclusively on the I-loader type.

- The input of the production line is illustrated on the left side of Figure 2.1. It comprises two components: the source, which generates unprocessed workpieces following a configurable distribution (in this instance, an uniform distribution), and a conveyor belt that transports the workpieces to the *Conveyor_In* station, where the loader can access and load the unprocessed workpieces.

- The right-hand side of Figure 2.1 illustrates the output of the production line. The output receives workpieces that have undergone all necessary processing

steps and transports them to the sink, thereby removing the workpiece from the simulation. The name of this output station is *Conveyor_Out*.

- A variable number of work centers may be positioned between the input and the output. Each work center comprises two machines as standard and is tasked with executing a specific work step, such as drilling or milling. These stations are named alphabetically starting with *Station_A* to *Station_F*. The processing time of the work step is defined by deterministic value. Furthermore, it is possible to simulate the occurrence of machine failures, which follow a stochastic behavior.

A process plan is specified for each workpiece type, describing the sequence of required work steps. Upon completion of these steps, the workpiece is considered finished and can be transported to the output.

A loader is capable of executing the following actions:

- The **drive** actions initiate the movement to the designated target station. The action name itself is the name of the target station.
- The **load** action initiates the loading process of a workpiece from the *Conveyor_In* or a work station into the gripper of the I-loader.
- The **unload** action initiates the unloading process of the loaded workpiece from the gripper into a work station or the *Conveyor_Out* station.
- The **wait** action allows the loader to await a change in the simulation environment, such as the completion of the processing of a workpiece at a workstation.

In contrast to conventional approaches, the digital twin simulation employs a continuous time frame, rather than discrete time steps. Moreover, the time required for the completion of individual actions varies. To illustrate, the duration of the *load* and *unload* action is a fixed value, whereas the duration of a drive action depends upon the initial location of the loader and the intended destination. The precise times are outlined in Chapter 7. In a production line that employs multiple gantry robots, it is typical for the loaders to complete their actions at disparate simulation times. In practice, it would be inefficient to wait for all loaders to complete their respective actions. Therefore, it is necessary for the loaders to receive new actions at different times, which results in the formation of an asynchronous system.

3

RELATED WORKS

Multi-Agent Reinforcement Learning (MARL) represents a research area that addresses the application of multiple agents that act in the same environment. In recent years, MARL has gained attention in the research community, as numerous real-world problems, such as cooperative robotics, autonomous driving, and business simulations, are characterized by the interaction of multiple agents. Multi-Agent (MA) approaches can be classified into three primary categories: competitive, cooperative, and mixed.

The competitive approach is characterized by scenarios in which multiple agents engage in a competitive pursuit, either to be the first to achieve a predefined goal or to accomplish their individual objectives. This approach is frequently observed in strategic games such as chess, Go [Sil+18], and StarCraft II [Vin+19], where the objective of the agents is to vanquish their opponent and thereby gain a superior position.

In contrast, the cooperative approach involves agents pursuing a common goal that can only be achieved through collaborative efforts. This type of MARL approach is frequently employed in applications that necessitate collective action, such as swarm robotics [Krn+24] or cooperative multiplayer computer games.

Mixed approaches integrate elements of both cooperative and competitive strategies. An illustrative example is autonomous driving [SSSS16; Zho+22], where each car strives to reach its destination as quickly as possible, while also coordinating with other cars to avoid collisions and ensure road safety.

In the context of the project discussed in this thesis, only cooperative MARL approaches are applicable, as all loaders and therefore their corresponding agents must work cooperatively to transport the workpieces through the production line.

In their book, Albrecht et al. [ACS24] provide an expanded classification of MARL approaches that illustrate their diversity and complexity. In addition to the three categories previously presented, MARL approaches can differ in the following dimensions. It should be noted that the respective main categories exclude certain characteristics of the dimensions.

- **Size:** The number of agents can be either constant or dynamic. A constant number of agents provides a stable framework, whereas a dynamic number of agents introduces greater complexity to the coordination process and necessitates the development of more adaptive strategies.
- **Knowledge:** The agents may vary in their knowledge of the environment and of each other. This may extend to include information regarding the action space, the reward function, and the state-transition probability.
- **Observability:** The observations made by the agents can be classified as either global or local. Global observations comprise all available information related

to the state of the environment and the actions of all agents. In contrast, local observations offer only a limited view of the environment for each individual agent. This behavior is also referred to as partial observability of the agents.

- **Reward**: The structure of rewards can be classified into three main categories: the competitive zero-sum reward, a shared reward, or individual rewards for each agent. The structure of rewards influences the development of strategies and the interactions between agents.
- **Objective:** The objective may be to develop a unified strategy that is advantageous for all agents or to optimize individual performance. Additionally, there is a distinction between whether only the final strategy learned or the evolving strategy during training is relevant.
- **Centralization and Communication:** The categorization of MARL approaches according to their training and execution behavior, as well as the communication patterns between the agents, is in accordance with the descriptions put forth by Gronauer and Diepold [GD22].
  - In *Centralized Training Centralized Execution* (CTCE) approaches, a common strategy is developed through centralized learning, with all agents utilizing the same strategy due to the direct exchange of information between the agents.
  - In *Centralized Training Decentralized Execution* (CTDE) approaches, the local experiences of all agents are shared and utilized for updating their policies. In contrast, during the selection of actions, each policy relies exclusively on the local observations of the respective agent, thereby entirely decentralizing the execution process.
  - In a *Distributed Training Decentralized Execution* (DTDE) approach, each agent gathers its own local experiences without any direct communication with the other agents. Their policies are trained exclusively on the basis of their individual experiences. Similarly, decisions are made exclusively on the basis of their local observations.

In consideration of the project setup utilized, it is possible to narrow down the specified dimensions. To illustrate, the number of agents is constant, as each agent is designed to control a single loader. It is highly improbable that the number of loaders would undergo a permanent alteration in such an application. Secondly, although the possible actions and the reward function are known, the state-transition probability is not known. In the context of the existing Single-Agent (SA) approach, it is essential that the agent is aware of the complete state of the environment. For the MARL approach, however, partial observations of local states are also a viable option for reducing the growth of the state space cardinality in more complex systems. Additionally, only shared or individual rewards can be considered for reward allocation, due to the cooperative objective. Similarly, the cooperative approach necessitates the learning of a common strategy, which must find a balance between the local strategies of the individual agents.

For the majority of problems that represent reality, a MARL approach with centralized training and execution is not a viable solution. In order to enable centralized execution, it is necessary to employ a controlling unit that must have a global view of the entire environment and all utilized worker agents. This results in the same issues as those encountered with the existing SA approach when a complex system is involved. Therefore, a decentralized execution approach is necessary. This is further reinforced by the fact that the actions of the individual agents must be executed independently due to the asynchronous nature of the simulation (see Chapter 2 and Chapter 5). Similarly, centralized training of the agent's policies based on the shared experiences is not applicable, as the local states of the agents may include different features of the global state, due to their intended partial observability.

In their work, Yu et al. [Yu+23] address the challenges of cooperative exploration by multiple robots in unknown regions, with a particular focus on improving exploration efficiency in real-world scenarios. The authors contend that traditional MARL methods assume synchronous action execution, which can be inefficient when the duration of the execution is different for each robot. The authors have proposed an asynchronous MARL approach, Asynchronous Coordination Explorer (ACE), which extends the multi-agent Proximal Policy Optimization (PPO) algorithm to handle asynchronous actions and incorporates action-delay randomization for better policy generalization.

The research of Gupta et al. [GEK17] examines the phenomenon of different cooperative Reinforcement Learning (RL) approaches in complex, partially observable environments, wherein the agents are not required to engage in explicit communication. They compare the three learning algorithms policy gradient, temporal-difference, and actor-critic. Testing reveals that among these methods, policy gradient methods demonstrate superior performance when used with feed-forward architectures. However, the environments used for evaluating the presented approaches were only step-based and involved synchronous action execution.

The work of Knrjiac et al. [Krn+24] addresses the order-picking problem in warehouses where mobile robots and human pickers must coordinate efficiently. The logistic problem is analogous to the project employed in this thesis, as heuristic methods require substantial engineering to optimize the control of the robots. The authors introduce hierarchical MARL algorithms, wherein a manager agent assigns objectives to worker agents, and both are co-trained to maximize a global metric like pick rate. These algorithms markedly enhance sample efficiency and pick rates in comparison to baseline MARL methods and industry heuristics across a range of warehouse scenarios. However, it uses a central manager agent which needs a global view of the environment.

The study conducted by Tampuu et al. [Tam+15] examines the interactions between two distinct Deep Q-Network (DQN) agents within the game of Pong. The study illustrates how modifying the reward scheme can prompt the emergence of both competitive and cooperative behaviors, whereby the cooperative setup aims to keep the ball in the game as long as possible. The research demonstrated the effectiveness of DQNs utilizing decentralized training in MA systems.

The presented works examine only specific aspects of the challenges and requirements inherent to the utilized project and do not consider the combination of all these elements. The combination consists of the following elements:

- A continuous RL task that does not have a finite state.
- The utilization of multiple agents that learn their local policy but work cooperatively.
- The employment of partial observability to reduce the complexity of the local states.
- The distributed and independent training of the heterogeneous agents.
- The decentralized execution of the selected action due to the asynchronous behavior of the environment.

The methodology used to solve this combination of requirements and challenges is presented in Chapter 5.

4

THEORETICAL FOUNDATIONS

This chapter provides an overview of the theoretical foundations of Reinforcement Learning (RL), with a particular focus on Q-Learning and Deep Q-Networks (DQNs). Additionally, a summary of the functionality of Neural Networks (NNs) is presented, as NNs are a crucial component of DQNs. These foundational concepts are essential to comprehend the subsequent Multi-Agent Reinforcement Learning (MARL) approach and its application, which will be discussed in subsequent chapters.

4.1 NEURAL NETWORKS

Artificial Neural Networks (ANNs), or NNs for brevity, represent a central component of machine learning and artificial intelligence. These models are based on the structure and functionality of the human brain, comprising interconnected nodes, known as neurons, that are organized in multiple layers. The neighboring layers are connected by establishing a link between each neuron of both layers. The strength of the signal transmitted from one neuron to the next is regulated by the adjustable weight of the connection between them. If the input layer is directly connected to the output layer, this is referred to as a single-layer NN. However, this type of network is not well-suited for mapping complex relationships, which is why additional layers, known as hidden layers, can be added between the input layer and the output layer. A multi-layered NN, as illustrated in Figure 4.1, is also referred to as Deep Neural Network (DNN) if it contains more than two hidden layers.

As illustrated in Figure 4.2, the perceptron represents the functional basis of a neuron in a NN. A perceptron receives multiple input values $x_0, x_1, .., x_n$, which are multiplied by the corresponding weights $w_0, w_1, .., w_n$ of the connection. Subsequently, the sum of the weighted input values, designated as $a = \sum_{i=0}^{n} w_i \times x_i$, is then passed on to an

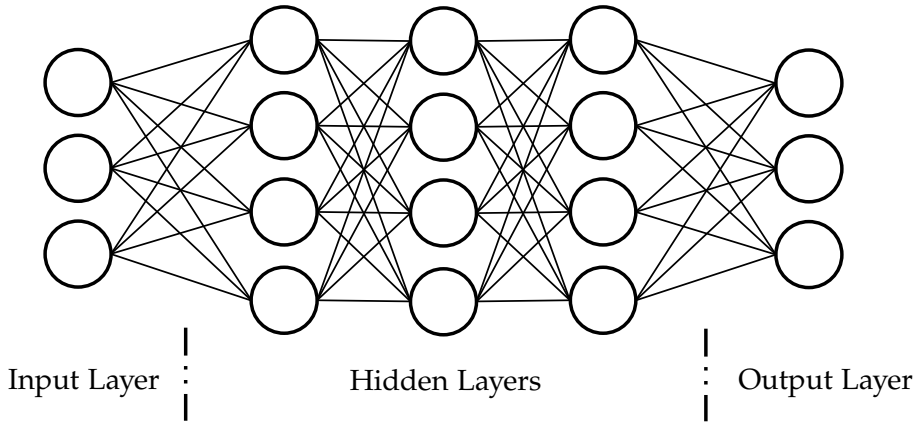

Figure 4.1: Illustration of a Deep Neural Network (DNN) with three hidden layers.

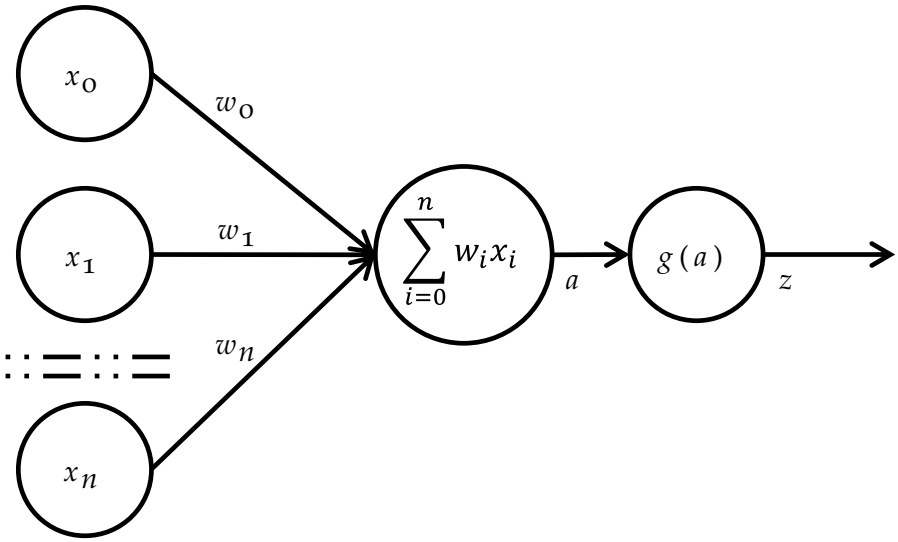

Figure 4.2: Perceptron with its components

activation function $g(a)$. The transformed output value z of the activation function represents the output of the perceptron. This output value is then utilized as input value for the subsequent perceptron. [DS13]

The output of the NN is dependent upon the specific weights, which must therefore be optimized through supervised training. The initial step is to perform the forward pass of the input data from the training data set. This implies that the input data is conveyed through each layer of the network, where each neuron calculates its value based on its input values and passes the result to all neurons of the next layer. Subsequently, the loss between the predicted values and the target values from the training data set is calculated. The loss function is used as an indicator of the quality of the network, and thus, the loss must be minimized to get better predictions. Subsequently, the calculated loss is propagated backward through the network in order to determine the gradients of the loss function in relation to each individual weight [RHW86]. The gradients indicate the extent to which each weight influences the overall loss of the network. Finally, all weights of the NN are updated using the gradient descent method, with the objective of minimizing the loss. These adjustments are made in an iterative process until the neural network achieves optimal performance.

In addition to their classification capabilities, NNs are also optimally suited as universal function approximators [DS13]. The theoretical basis for this capability is the *Universal Approximation Theorem*, which was formulated by Hornik et al. [HSW89]. This theorem demonstrates that any function of finite dimension can be approximated by a multi-layer feedforward NN. The degree of accuracy achieved in this approximation is dependent upon the structure of the NN, particularly the number and configuration of layers and neurons, as well as the quality of the training data. The successful approximation of non-linear target functions requires the use of non-linear activation functions, such as the Rectified Linear Unit (ReLU) function, in the hidden layers. In the absence of non-linear activation functions, the network is constrained to linear relationships. Consequently, the selection of an appropriate activation function is important in order to utilize the

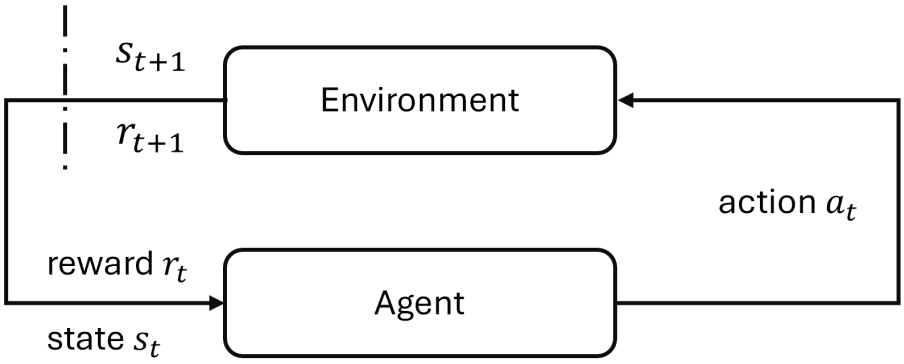

Figure 4.3: Interactions between Agent and Environment.

capabilities of NNs fully. Furthermore, networks comprising multiple output neurons allow for the approximation of vector-based functions and the mapping of intricate relationships in multi-dimensional spaces.

4.2 REINFORCEMENT LEARNING

The objective of machine learning is to achieve the most accurate prediction or classification possible based on input data. The learning algorithms that have been developed can be classified into three main categories: supervised, unsupervised, and reinforcement learning [Bis06]. The categories differ in terms of the manner in which the learning algorithm operates. In supervised learning, both the training data and the desired target values are available. In contrast, an unsupervised learning procedure is required to recognize correlations using only the training data, without the benefit of target values. In contrast, reinforcement learning does not require the initial provision of training data and target values, as this data is generated during the training phase.

Figure 4.3 provides a straightforward representation of the structure of a RL problem. The environment presents the mapping of the problem and all potential states. In the initial stage, the RL agent receives the current state, designated as s_t, of the environment at time t. Based on the state, the agent then selects an action, designated as a_t, which is then transmitted to the environment. Subsequently, the environment determines the following state, designated as s_{t+1}, and calculates the reward for the resulting state-action-state transition, represented as (s_t, a_t, s_{t+1}). The new state and the associated reward for the preceding action are communicated to the agent, who then selects the subsequent action based on the new state. This process is repeated until either a final state of the environment is reached or, in the case of a continuous RL problem, the process continues until an artificial endpoint is reached. This may be, for example, until a predefined number of time steps have been completed. The objective of RL is to instruct the agent in a strategy that will result in the selection of the optimal action a_t in a given state s_t, thereby maximizing the total reward received by the agent over the course of the episode.

The following explanations and formulas follow those used by Sutton and Barto in their book, *Reinforcement Learning: An Introduction* [SB18].

4.2.1 *Definitions and Explanations*

The property of RL problems that decisions are contingent upon the current state implies that the subsequent state is also contingent upon the current state. This indicates that RL problems satisfy the Markov property, which characterizes this dependency. Consequently, an RL problem can be formalized as a Markov Decision Process (MDP) based on the following tuple $MDP = (S, A, R, p)$, where S is the set of possible states of the environment, A is the set of possible actions and R is the set of rewards [SB18]. The state transition function is defined as follows: $p(s', r|s, a) = P(S_t = s', R_t = r|S_{t-1} = s, A_{t-1} = a)$. This specifies the probability with which the system transitions to a certain subsequent state, thereby describing the dynamics of the MDP.

As previously stated, the objective of RL is for the agent to learn to select actions in a way that maximizes the reward. The return G_t at time t is defined as the cumulative sum of future rewards (see Equation 4.1):

$$G_t = \begin{cases} \sum_{k=0}^{T} R_{t+k+1} & \text{for problem with a finite state} \\ \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} & \text{for continuous problems} \end{cases} \tag{4.1}$$

In the event that the RL problem lacks a clearly defined endpoint by mapping a continuous problem, the discontinuation rate $0 \leq \gamma \leq 1$ is employed to weight future rewards. If $\gamma = 0$, only the next reward is considered, resulting in a myopic decision-making process. Conversely, if $\gamma = 1$, the reward grows to infinity, which is also problematic. Therefore, it is advisable to select a discount factor $\gamma < 1$ to ensure that the reward converges to a single value.

The behavior of the agent is contingent upon its current so-called policy. The agent's policy, denoted by π, is defined as the probability of selecting action A_t in state S_t (see Equation 4.2).

$$\pi(S_t) = P(A_t|S_t) \tag{4.2}$$

The policy π allows the evaluation of the value of a given state S_t through the use of the value function (Equation 4.3).

$$v_\pi(s) = \mathbb{E}_\pi[G_t|S_t = s] \tag{4.3}$$
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma\, G_{t+1}|S_t = s] \tag{4.4}$$
$$v_\pi(s) = \mathbb{E}_\pi[R_{t+1} + \gamma\, v_\pi(s_{t+1})|S_t = s] \tag{4.5}$$

The value function defines the value of a state by the expected return, that can be achieved from that state. Upon inserting the definition of the return (Equation 4.1), it becomes evident that this expected value considers all potential subsequent states of the current state in a recursive manner, as illustrated in Equation 4.4 and Equation 4.5.

In addition to evaluating a state alone, the value of an action in that given state can also be evaluated using the action-value function or Q-function for short. The Q-function

is defined as the expected return G_t of an action a based on the current state s and the current policy pi (see Equation 4.6). The future reward is not defined by the value of the subsequent state s_{t+1}, but by the action a' that has the highest Q-value in that state.

$$q_\pi(s,a) = \mathbb{E}_\pi[G_t|S_t = s, A_t = a] \tag{4.6}$$

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma \max_{a'} q_\pi(s_{t+1}, a')|S_t = s, A_t = a] \tag{4.7}$$

By transforming the value function into a weighted sum of all possibilities, the *Bellman equation* (Equation 4.8) is obtained. The value of a given state s can be described in words as follows: For each action $a \in A$, the value of all potential subsequent states is determined. This value is weighted with the policy $\pi(a|s)$, which is the probability that a specific action a is selected in the given state s. The value of all subsequent states is then determined by the expected return, which is weighted by the transition probability $p(s', r|s, a)$ into that respective state s'.

$$v_\pi(s) = \sum_{a\in A} \pi(a|s) \sum_{r\in R, s'\in S} [r + \gamma\, v_\pi(s')] * p(s', r|s, a) \tag{4.8}$$

The objective of RL is to identify an optimal policy, denoted as π_*. A policy π is considered superior to another policy π' if for all state $s \in S$ applies that $v_\pi(s) \geq v_{\pi'}(s)$. It is possible that multiple optimal policies may exist. The optimal value function resulting from the optimal policy is defined in Equation 4.9. The same applies to the optimal action-value function $q_*(s,a)$.

$$v_*(s) = \max_\pi v_\pi(s) \qquad \text{for all} \quad s \in S \tag{4.9}$$

In assuming the optimal value function is present, the value of a state must be equivalent to the value of the optimal action from the state. The resulting *Bellman optimality equation* (Equation 4.10) demonstrates that the optimal value function can be determined without the knowledge of the optimal policy.

$$\begin{aligned} v_*(s) &= \max_{a\in A(s)} q_{\pi_*}(s,a) \\ &= \max_a \mathbb{E}_{\pi_*}[G_t|S_t = s, A_t = a] \\ &= \max_a \mathbb{E}[R_{t+1} + \gamma\, v_*(S_{t+1})|S_t = s, A_t = a] \end{aligned} \tag{4.10}$$

However, since neither the optimal value function nor the optimal policy is provided for the majority of RL problems, these must be determined approximately using the learning procedure. Solving the Bellman optimality equation for each state is only feasible for problems with very small state spaces (e.g. with Tabular Q-Learning). For problems with large state spaces, the value or Q-function must be approximated. There are numerous learning methods available for this approximation. However, this paper will focus on the Q-learning method used with DQNs.

4.2.2 *Temporal-Difference Learning*

The Temporal-Difference (TD) learning approach is one of the most widely used learning methods and forms the basis for Q-learning, which is explained in the following section. TD learning is itself a combination based on the ideas of the Monte Carlo methods and dynamic programming. As a model-free learning technique, TD learning employs an empirical trial-and-error process to identify an optimal policy over numerous time steps and episodes.

Both the TD and Monte Carlo methods rely on accumulated experience to refine the approximation of the value function v_π. However, the Monte Carlo method is only capable of performing the update after the episode has been completed, as the final return G_t is necessary for the update. The update function of the Monte Carlo method is delineated in Equation 4.11, which also employs the learning rate α to regulate the impact of the update.

$$V(S_t) \leftarrow V(S_t) + \alpha[G_t - V(S_t)] \tag{4.11}$$

$$V(S_t) \leftarrow \max_{A_t} \sum_{S_{t+1}, R_{t+1}} [R_{t+1} + \gamma\, V(S_{t+1})] * p(S_t{+}1, R_{t+1} | S_t, A_t) \tag{4.12}$$

The dynamic programming approach offers the advantage of enabling the value function approximation to be updated in the subsequent time step, $t+1$. However, the dynamic programming approach requires the transition probability, as defined in Equation 4.12, which is often unavailable for a multitude of RL problems.

The TD learning approach facilitates the incorporation of both approaches, enabling the update to occur in the subsequent time step and with an unidentified transition probability. To accomplish this, the error function is adapted so that the split return is used for the TD-error, which is defined as follows: $R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)$. The complete update function for the TD learning approach is delineated in Equation 4.13.

$$V(S_t) \leftarrow V(S_t) + \alpha[R_{t+1} + \gamma\, V(S_{t+1}) - V(S_t)] \tag{4.13}$$

4.2.3 *Q-Learning*

The Q-learning approach represents an extension of the previously discussed TD learning approach. Therefore, the TD learn algorithm is transferred from the value function to the action-value function or the Q-function. The update of the approximation Q for the Q-function q_* is illustrated in Equation 4.14 [WD92].

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma\, \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{4.14}$$

As indicated by the equation, a greedy selection is employed for the target value $R_{t+1} + \gamma\, \max_a Q(S_{t+1}, a) - Q(S_t, A_t)$, whereby the action with the maximum Q-value in the subsequent state is selected. This selection of the action is not based on the existing policy π, and thus Q-learning can be classified as an off-policy procedure.

As a consequence of the Q-function assigning a value to each state-action combination, it is possible to store all of these values in a table, which is referred to as a Q-table. It should be noted, however, that a restriction is in place that both the state and action spaces are discrete. Algorithm 1 illustrates the learning process. At the beginning of the learning process, the learning rate α must be defined, and the Q-table must be initialized with arbitrary values for each action-state combination. In order to begin a new episode, it is required that the environment state S_t is reset to its initial state S_0. Subsequently, the following steps are repeated until a terminal state is reached in the environment. Firstly, the action A_t is selected either based on the Q-function or randomly, if the ϵ-greedy approach is utilized. The ϵ-greedy approach is designed to support the learning process by enabling the exploration of states through random selection. The selected action is then executed and the subsequent state S_{t+1} and the assigned reward R_{t+1} are observed. Subsequently, the corresponding Q-value in the Q-table is updated based on the observed data. Finally, the current state is updated, and the next loop begins.

Algorithm 1 Q-Learning algorithm [SB18]

Require: Algorithm Parameters: learning rate $\alpha \in (0, 1]$, ϵ-greedy value
Require: $Q(s, a)$ initialized, for all $s \in S, a \in A$
1: **for** each episode **do**
2: Initialize state $S_t = S_0$
3: **while** $S_t \neq$ *terminal* **do**
4: Select A_t for S_t based on policy of current Q or ϵ-greedy approach
5: Execute A_t and observe reward R_{t+1} and next state S_{t+1}
6: $Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$
7: $S_t \leftarrow S_{t+1}$
8: **end while**
9: **end for**

4.2.4 *Deep-Q-Networks*

The issue of RL problems with very large state spaces has the potential to cause difficulties for learning approaches such as the Q-tabular learning method that has been presented above. As a consequence of the rising complexity of the environment state, its dimension expands at an exponential rate, such that the states reached during training constitute a mere fraction of the total. The phenomenon was initially referred to as the *Curse of Dimensionality* by Richard Bellman [BBC57]. For this reason, methods that approximate the Q-function are employed in favor of tabular approaches. As previously stated, neural networks are highly effective at approximating arbitrary functions. The combination with Deep Learning approaches is called Deep Reinforcement Learning.

The so-called Deep Q-Network (DQN) approach employs Deep Neural Network to approximate the action-value function as a non-linear approximation [Mni+15]. The DQN is tasked with predicting the Q-values for all potential actions, based on the state input to the NN. The values of the neurons in the output layer are interpreted

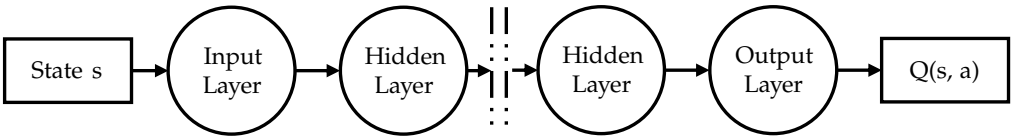

Figure 4.4: Structure of a Deep Q-Network (DQN) for Q-value Approximation

as the Q-values, which is illustrated in Figure 4.4. Like other NNs, the DQN is trained through backpropagation of the error made (see Section 4.1). The squared error between the predicted value $Q(S_t, A_t; \theta)$ and the target value $R_{t+1} + \gamma \max_a Q(S_{t+1}, a; \theta)$ is typically employed as the loss function (see Equation 4.15). The parameters of the NN are collectively symbolized by the symbol θ.

$$L_e(\theta) = \left(Q(S_t, A_t; \theta) - \left[R_{t+1} + \gamma \max_A Q(S_{t+1}, A; \theta) \right] \right)^2 \tag{4.15}$$

A significant challenge inherent to the DQN approach is the necessity to predict the Q-values of all actions in every conceivable state with the same Network parameters. Consequently, the structure of the DQN must be tailored to accommodate the complexity of the state space.

Another issue that arises is that the approximation function is used both for the prediction of the predicted Q-value and for the target value. In order to prevent correlation between these values, the DQN approach employs the use of two distinct NNs, namely the Q-network and the target network. The Q-network is described by the function $Q(S_t, A_t; \theta)$, whereas the target Q-network is described by the function $Q(S_t, A_t; \theta^-)$. The prediction network is used to predict the Q-value of an action in a given state. The target network is employed to evaluate the selected action by greedily selecting the action that maximizes the Q-value in the subsequent state S_{t+1}, thereby identifying the optimal action. It is necessary that both networks use the same structure as only the weights of the Q-network are updated via Q-Learning. The weights of the target network are updated only periodically, at intervals of C steps, by synchronizing with the weights of the Q-network. The use of two networks also serves to address another issue, namely that minor adjustments to the weights may result in significant alterations to the current policy. As the weights of the target network are updated with a delay, the target values remain constant until the next update, thereby counteracting rapid and unintended policy changes in such circumstances.

Additionally, correlations may occur in the sequence of collected experiences due to the limited adjustments made to the Q-function approximation. To address this issue, a solution has been proposed in the form of replay memories, which involve the temporary storage of collected experiences [Lin92]. An experience is defined as a tuple comprising four elements: S_t, A_t, R_{t+1}, and S_{t+1}. By randomly selecting a mini-batch of experiences from the replay memory, it is possible to avoid any correlations in the data that is used to train the DQN. The decoupling of the actual DQN training from the complete learning process allows for straightforward control over the frequency and number of experiences used in the training of the Q-network.

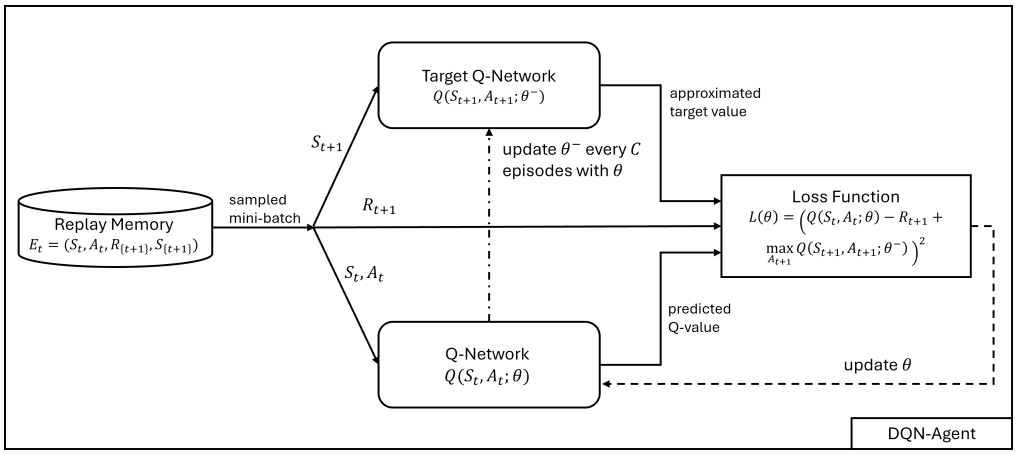

Figure 4.5: Training Process of a DQN Agent.

A further disadvantage is that the replay memory is initially empty and must therefore be populated. For this reason, it is reasonable to utilize an exploration phase of the state space using a 100 % ϵ-greedy approach, for example, until the requisite number of experiences are available in the replay memory. Subsequently, if a sufficient quantity of experience is available, the ϵ value can be adjusted so that the newly acquired experiences contain decisions made by the DQN. To prevent older and probably suboptimal experiences, which were generated using outdated policies, in the replay memory from subsequently influencing the approximation, it is advisable to limit the size of the replay memory. Once the maximum size has been reached, the oldest experiences are removed from the replay memory. It is crucial to ensure that a correspondingly large number of experiences is permitted, in order to guarantee an adequate level of variance.

Figure 4.5 illustrates the structure and connection of the aforementioned components of a DQN agent. The training process of the DQN is illustrated in Algorithm 2 in a step-by-step manner. The process begins with the initialization of the replay memory D and the construction of the Q-network and the target network. The initial weights employed for both networks are identical. Each episode begins with the reset of the environment to its initial state S_0. In each time step t, the action A_t is selected either by the current policy based on the Q-value approximation or by a random selection from the action space, when the ϵ-greedy approach is used. Subsequently, the action is executed, and the assigned reward R_{t+1} and the subsequent State S_{t+1} are observed. The observed data is stored as experience tuple $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ in the replay memory. The training of the weights of the Q-network is illustrated in lines 7 to 13 in Algorithm 2. In this example the training is executed in each time step; however, it is also possible to train less frequently. Initially, a mini-batch of experiences is randomly selected from the replay memory. Subsequently, for each experience within the mini-batch, the Q-value for the current action-state pair is predicted using the Q-network. With the target network the target value $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ is predicted. Afterward, the loss is calculated between the target and the predicted value. In this example, the squared error is employed. Subsequently, the total loss of all experiences within the mini-batch is used to update the weights θ of the Q-network by utilizing the backpropagation method.

Finally, the weights of the target network are updated periodically at regular intervals, with a period of C steps, by copying the current weights of the Q-network.

Algorithm 2 Deep-Q-Learning algorithm

Require: Initialize replay memory D
Require: Initialize Q-Network and target-network with the same weights θ, θ^-
for each episode **do**
  Initialize state $S_t = S_0$
  while $S_t \neq terminal$ **do**
    Select A_t for S_t based on policy of current Q or ϵ-greedy approach
    Execute A_t and observe reward R_{t+1} and next state S_{t+1}
    Store experience $e_t = (S_t, A_t, R_{t+1}, S_{t+1})$ in replay memory D
    Sample random mini-batch from replay memory
    for each $e \in$ mini-batch **do**
      Predict Q-value for experience and receive $Q(S_t, A_t; \theta)$
      Approximate target Q-value $Y_t = R_{t+1} + \gamma \max_A Q(S_{t+1}, A; \theta^-)$
      Compute loss $L_e(\theta) = (Q(S_t, A_t; \theta) - Y_t)^2$
    end for
    Update θ by back-propagating total loss of mini-batch in Q-network
    Every C steps set $\theta^- \leftarrow \theta$
    $S_t \leftarrow S_{t+1}$
  end while
end for

5

METHODOLOGY

This chapter presents the methodology for implementing an independent Multi-Agent Reinforcement Learning (MARL) approach using Deep Q-Network (DQN) for the dynamic scheduling of gantry robots. The initial section delineates the fundamental functionality and communication of the Reinforcement Learning (RL) agent with the simulation. Subsequently, the shortcomings of the Single-Agent Reinforcement Learning (SARL) approach for production lines with a more complex layout are elucidated. Finally, the methodology for the MARL approach is presented and explained.

5.1 SIMULATION

The simulation returns a simulation state that contains a multitude of features for each individual component described in Chapter 2. The state utilized for RL is a configurable subset of features that are described in Table 5.1. The *Conveyor_In* and *Conveyor_Out* stations are also managed as normal stations but only the *finished* feature from the *Conveyor_In* is used. This feature is renamed to *available* in the table in order to improve comprehensibility. No features from the *Conveyor_Out* station are needed as it will always accept workpieces and is incapable of failure. In the case that multiple loaders are used the possible values for the *action* feature of the loaders the value "idle" is added, which is used for the requesting loader, but can not actively be executed.

5.1.1 *Simulation Process*

In the case of production lines comprising a single loader, the simulation will transmit a request to the agent upon the completion of previously received action a_{t-1}. The current state s_t transmitted within the request comprises, in comparison to the previous state s_{t-1}, both "active" changes resulting from the last action a_{t-1} itself, and "passive" modifications that occurred independently to this action. An example of an "active" change would be a drive action or the loading of a workpiece. For example, a "passive" state modification includes the completion of a workpiece processing cycle at a work station or a failure of a machine. The *wait* action is an exception to this rule; the waiting of the loader gets canceled when a change in the simulation state is detected. Previous suboptimal actions may have caused a simulation state in which no more passive changes can occur. This is the reason behind the simulation's automated cancellation of waiting actions after a specified elapsed time.

A training session is constituted of multiple episodes that progress at an accelerated pace. Each episode starts in a defined state, wherein all machines are unproductive. The conclusion of an episode is triggered after a predefined simulated time, such as

Table 5.1: Features of the simulation state used in the Reinforcement Learning approaches.

Feature	Description
System State	General features
Simulation Time	Time step of the request
Episode Throughput	A counter of finished processed and delivered workpieces in the running episode
Requesting Loader	Id of the requesting loader
Reward Last Action	Reward type for the last action of the requesting loader
Loader State	Feature of each loader
Position	Current location (==station) of the loader
Gripper Object	Data of loaded object in the gripper of the I-loader
Action	Current executed action
Blocked	Flag when loader can't complete action because another loader blocks its way
Conveyor_In State	
Available	Flag if a new unprocessed workpiece is available
Station State	Features of each work station
Occupied	Flag if a workpiece is processed
Finished	Flag if processing has finished
Failed	Flag if the machine has a failure

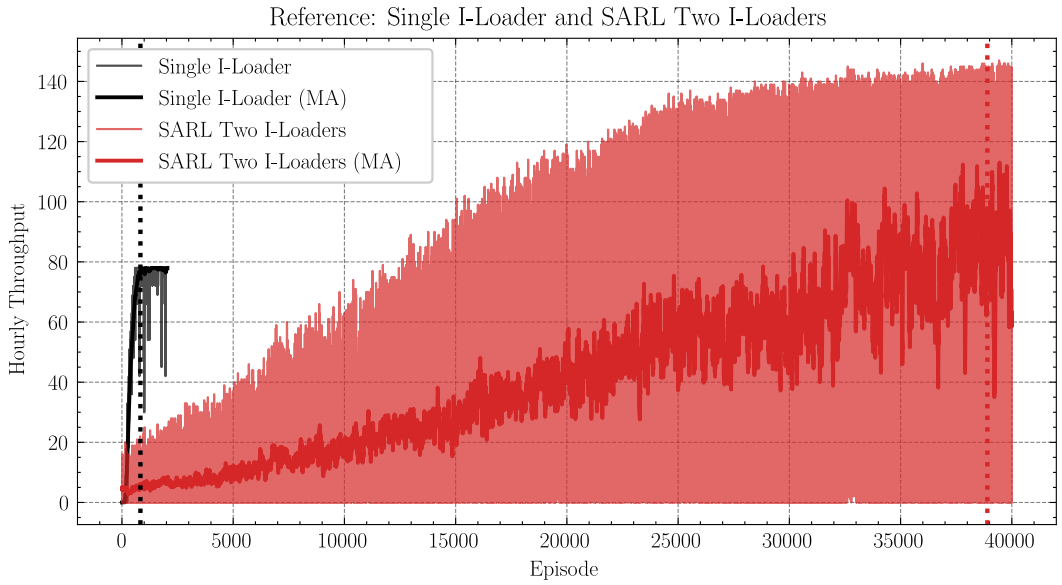

Figure 5.1: Comparison of the training progress between a single I-loader and two I-Loaders production line using the SARL approach.

20 minutes. The required simulated time for optimal training depends upon various factors, including the number of work steps and the processing time of the machines.

Furthermore, the simulation is also capable of controlling multiple loaders in an asynchronous manner. This implies that the loaders can execute their actions independently and request new actions from the agent at different simulation times. In the event of multiple loaders transmitting their requests simultaneously, the requests are processed in a predetermined sequence, with the leftmost loader receiving priority. This occurs at the beginning of an episode, at simulation time $t_0 = 0\,\mathrm{s}$, or if one loader is currently waiting and another loader completes its action, thereby canceling the waiting. This results in the phenomenon whereby the global state is updated prior to the transmission of the next loader request, creating two different global states for the same simulation time. This is not a significant issue, as the loader, which transmits the request, is a feature of the global state in the SARL approach. Additionally, the training of the DQN is independent of the absolute simulation time.

5.2 PROBLEM FORMULATION

The implementation of the simulation and of the existing SARL approach is works well for production lines with only one single gantry loader and only a limited number of machines. However, if the produced workpiece requires more process steps and therefore more machines on the production line, it is possible that using only one loader can result in a bottleneck, i. e. the loader cannot transport each workpiece fast enough to its next station. In practice, the solution is to add more loaders to the production line.

However, this introduces further challenges for the RL algorithm. The required training duration or rather the number of needed episodes for the Single-Agent (SA) approach increases significantly as seen in Figure 5.1. It is also noticeable that the training with

the more complex production line setup is considerably less stable. The challenges of the SARL approach will be explained in the following sections.

5.2.1 *Random Request Sequence*

The simulation is capable of controlling multiple loaders in an asynchronous manner. Nevertheless, the requests to the agent are transmitted in a synchronous manner, allowing only the request for a new action for a single loader at any given time. Consequently, the identifier of the requesting loader must be included in the request, so that the agent is aware of which loader the action should be selected for. Depending on the execution duration of the selected action it may result in the situation where one or multiple requests for the other loader are handled until the initial loader sends its next request. The SARL approach uses always the state of the subsequent request at time step $t+1$ as the next state s_{t+1} in the replay memory entry. However, due to the potential for interim requests from other loaders, the next state s_{t+1} may be one in which the previously chosen action a_t is fully executed. Alternatively, it may be a state where the action a_t is still being executed when the next request is triggered from another loader. This behavior is illustrated in Table 5.2.

This is a problem for the SARL approach, as the entries in the replay memory can be inconsistent. Looking at this problem from a theoretical point of view, the set of next states for each state-action-pair contains many different states. Only a few of them are states, where the last chosen action is fully executed (with few variations based on passive environment changes). The majority of the states are states, where the last chosen action is still being executed. During the training of the DQN, the future Q-values are calculated based on these states (see Section 4.2.4), which leads to a high variance in the Q-values and therefore to a slow learning process.

5.2.2 *Reward Distribution*

As an additional consequence of the issue outlined in Section 5.2.1, the reward for the action a_t must be calculated by the SARL approach in time step t. This is necessary because the simulation is designed in such a way that the reward for an action is only returned when the same loader sends its next request. This may occur in the next time step $t+1$ or any following time step $t_i \quad \text{for} \quad i > 1$. For the training of the DQN the reward it is essential that the reward is assigned to the correct state s_t and action a_t.

As a consequence of the fact that only the last loader can receive the highest reward for delivering a finished workpiece at the *Conveyor_Out* station, a further problem has emerged. This issue may influence the learned strategy or, at the very least, the training duration. This issue arises because the agent may attempt to position the first loader in close proximity to the *Conveyor_Out* station, as the expected return may be higher in these states. This would result in unnecessarily lengthy journeys to reposition the first loader when, for example, the processing of a workpiece in *Station_A* has finished.

Table 5.2: Generated experience by the SARL approaches for a random sequence of requesting loader. The "Completed Action" rows indicate whether the replay memory entry has a new state s_{t+1}, where the action a_t was fully executed.

Step	0	1	2	3	4
Requesting Loader	L1	L2	L1	L1	L2
State	s_0	s_1	s_2	s_3	s_4
Action	a_0	a_1	a_2	a_3	a_4
Reward	r_0	r_1	r_2	r_3	r_4
SARL					
Experience	-	(s_0, a_0, r_0, s_1)	(s_1, a_1, r_1, s_2)	(s_2, a_2, r_2, s_3)	(s_3, a_3, r_3, s_4)
Completed Action	-	✗	✗	✓	✗

Step	5	6	7	8
Requesting Loader	L2	L2	L1	L2
State	s_5	s_6	s_7	s_8
Action	a_5	a_6	a_7	a_8
Reward	r_5	r_6	r_7	r_8
SARL				
Experience	(s_4, a_4, r_4, s_5)	(s_5, a_5, r_5, s_6)	(s_6, a_6, r_6, s_7)	(s_7, a_7, r_7, s_8)
Completed Action	✓	✓	✗	✗

Table 5.3: Comparison of the state space cardinalities for production lines with one and two operating I-Loaders.
A detailed overview of features for the used states is illustrated in Table A.1.

Loader Setup	State Space Cardinality
One I-Loader	3.3×10^5
Two I-Loaders	7.8×10^9

5.2.3 *Loader Interactions*

For production lines with multiple gantry loaders it is essential to avoid collisions between loaders. A collision occurs in the event of either a loader driving into another stationary loader or two loaders driving in opposite directions toward each other. For example, one loader is located at *Station_A* and has decided to drive to *Station_D*, but at the same time, the other loader is already driving from *Station_E* to *Station_B*.

On one hand, a collision is addressed by the simulation by adding the blocked state to the loader. If a loader is driving to a station, that is already occupied by another loader, the loader will wait in front of that station until the other loader has driven away. This special waiting is signalized by the blocked status of the loader, which must be considered by the SARL agent. On the other hand, the SARL approach restricts the possible actions for the requesting loader, by removing all drive actions to stations temporarily from the action space, that are located behind the blocking loader.

5.2.4 *State Space Cardinality*

In the SARL approach the state space must represent the complete environment, as the agent must be able to make decisions for the complete environment. This results in a state space that grows rapidly, when the number of stations and loaders increases. In practice, the addition of a second loader should improve the productivity of the production line as the loaders can work in parallel. However, the state space grows considerably with the introduction of a second loader, as the state space must contain all relevant information about the second loader.

Example
We consider a production line with three work centers, with each two machines, and simulated machine failures. Table 5.3 illustrates the cardinalities of the state spaces for one and two I-loaders in this production line. In the case that a single I-loader operates within this production line, the state space has a cardinality of approximately 3.3×10^5 different states. The introduction of a second I-loader increases the state spaces to about 7.8×10^9 states. Table 5.4 illustrates the additional features that are required in the state and the number of potential values associated with each feature. By introducing the second loader the state space cardinality grows by a factor of 23 716. A detailed

Table 5.4: Composition of the growth factor for the state space cardinality, based on the utilization of a second loader. The potential values for the position of loader 1 are reduced from eight to seven, as loader 1 is no longer capable of driving to the *Conveyor_Out* station. This results in a reduction in the growth factor from 27 104 to 23 716.

Additional Features	**# Values**
Requesting Loader ID	2
First Loader	
Position (reduced)	$8 \rightarrow 7$
Current Action	11
Blocked Status	2
Second Loader	
Position	7
Gripper Status	4
Current Action	11
Blocked Status	2
Growth Factor	$27\,104 \rightarrow 23\,716$

overview of the used features and their number of potential values are illustrated in Table A.1.

In practice, production lines typically comprise a considerably larger number of machines and loaders than the system described in the previous example. This provides an insight into the immense scale of the state space of a real system, should all components be mapped together. The growth is amplified by the addition of a single machine, which not only increases the cardinality by the value of 4 for the machine's features but also expands the potential values for the *position* and *current action* feature of each loader. Consequently, the growth rate for each added station and loader is considerably higher, exhibiting an exponential pattern.

5.3 DECENTRALIZED AND INDEPENDENT COOPERATIVE MULTI-AGENTS

This thesis presents a new approach that introduces multiple agents, each is responsible for controlling a single loader. It is crucial for the agents to collaborate in a cooperative manner to accomplish the overarching objective of maximizing the number of finished workpieces produced in the simulated production line.

In the SARL approach, the single agent is required to observe the global state space, which consequently results in an exponential growth in the state space S when additional machines or loaders are integrated into the production line (see Section 5.2.4). Nevertheless, in the MARL approach with multiple independent agents, the use of partial observability can offer a potential advantage. Partial observation describes a scenario in

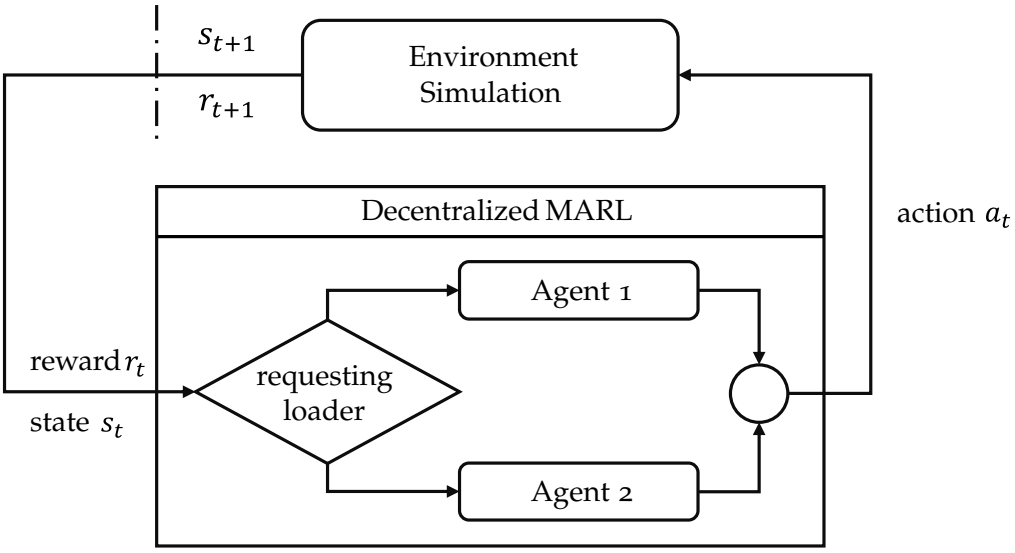

Figure 5.2: Structure of the RL problem with two decentralized Agents.

which the agents must make their decision based on a subset of the features from the entire environment state S. Each agent i has its own so-called local state space S_i.

$$S_i \subseteq S \quad \text{for each agent } i$$

In this project setup also a local action space A_i is used for each agent i:

$$A_i \subseteq A \quad \text{for each agent } i$$

The implementation of a local action space leads to a reduction in the number of output neurons from the DQN, which consequently reduces the number of trainable parameters. In considering the structure of the production line in this project with the specified agent count N, it becomes evident that only the last loader is able to proceed towards the *Conveyor_Out* station. This allows for the removal of the *Conveyor_Out* action from every other local action space A_i, for $i = 1, \ldots, (N-1)$. A similar principle applies to the *Conveyor_In* action, as only the first loader can drive to the *Conveyor_In* station. Furthermore, only the local state space S_1 of the first agent must contain the state features for the *Conveyor_In* station.

The utilization of local state and action spaces for each agent necessitates the establishment of a distinct Q-network and target network for each agent. This is necessary because, on the one hand, the input layer maps the features of the local states, and on the other, the output layer maps the local action space. Consequently, each agent must also save its collected experiences in its own replay memory and be trained independently.

Figure 5.2 illustrates two agents and their connections to the environment. Depending on the value of the feature "requesting loader" (see Section 5.1) the global state s_t and the reward r_t is passed to the corresponding agent. The passed reward r_t is the reward for the last action that the same loader has executed beforehand as the simulation has cached the reward until the same loader sends the next request.

As illustrated in Table 5.5 the distribution ensures that only entries in the individual replay memories are stored where the previous action a_{t_i} of the same agent was fully executed. Any other actions that were processed by other agents and executed by their associated loaders can be seen as "passive" updates (see Section 5.1.1) in the state. This will enhance the learning efficiency of the DQN, as each entry in the replay memory will be consistent. To illustrate, the initial state s_t where the loader is positioned at *Station_A* and the selected action a_t is the drive to *Station_C*, the position of that loader will invariably be *Station_C* in the next state s_{t+1}.

5.3.1 *Work Area Restriction*

A further reduction of the local state and action spaces can be achieved by limiting the operational area of the loaders. In practice, it is unlikely that each loader will have access to all machines in a production line due to the necessity of connecting the mobile loaders to power and network, which is often done via drag chains. Moreover, the necessity for different types of grippers for the handling of intermediate workpieces must be considered. It is also assumed that the RL algorithm will partition the production line into relatively equal sections with the aim of minimizing travel times and distances.

In order to prevent collisions between loaders utilizing the SARL approach, it was essential to include all information of all loaders into the state space. This would inevitably lead to an exponential expansion of the state space with the addition of numerous loaders to the production line. From the perspective of a loader, it is only sufficient to know the state of the neighboring loaders, as these alone represent a potential risk of collision.

Table 5.6 illustrates based on the previous statements the necessary features of an agent's local state spaces S_i. Depending on the production line setup, the number of loaders, work steps, and agent responsibility, the number of needed features varies. The cardinalities of the state spaces based on different approaches and restrictions are illustrated in Table 5.7. All values are based on the same production line setup with three work steps which can each be processed at two stations. As previously stated (see Section 5.2), the introduction of a second loader to the production line significantly increases the state space cardinality when the SARL approach is utilized. Two different configurations are presented for the MARL approach, which distinguishes in terms of the responsibilities assigned to the loaders. In the case of "full stations responsibility", both loaders are capable of driving to all workstations (each conveyor station can still be reached by only one loader). In the alternative configuration, the production line is divided into two sections. Consequently, the first loader is capable of driving from the *Conveyor_In* station to *Station_D*, while the second loader can drive between *Station_C* and the *Conveyor_Out* station. This reduction in responsibility mitigates the risk collisions between loaders only to *Station_C* and *Station_D*. Furthermore, it reduces the cardinality of both agents' local state spaces to a value comparable to that of the single I-loader SARL setup.

Table 5.5: Comparison of the generated experience by the SARL and MARL approaches, based on the same sequence of requesting loaders. The "Completed Action" rows indicate whether the replay memory entry has a new state s_{t+1}, where the action a_t was fully executed.

Step	0	1	2	3	4
Requesting Loader	L1	L2	L1	L1	L2
State	s_0	s_1	s_2	s_3	s_4
Action	a_0	a_1	a_2	a_3	a_4
Reward	r_0	r_1	r_2	r_3	r_4
SARL					
Experience	-	(s_0, a_0, r_0, s_1)	(s_1, a_1, r_1, s_2)	(s_2, a_2, r_2, s_3)	(s_3, a_3, r_3, s_4)
Completed Action	-	✗	✗	✓	✗
MARL					
Agent 1 Step	0	-	1	2	-
Agent 2 Step		0	-	-	1
Agent 1 Experience	-	-	(s_0, a_0, r_2, s_2)	(s_2, a_2, r_3, s_3)	-
Agent 2 Experience	-	-	-	-	(s_1, a_1, r_4, s_4)
Completed Action	-	-	✓	✓	✓

Step	5	6	7	8
Requesting Loader	L2	L2	L1	L2
State	s_5	s_6	s_7	s_8
Action	a_5	a_6	a_7	a_8
Reward	r_5	r_6	r_7	r_8
SARL				
Experience	(s_4, a_4, r_4, s_5)	(s_5, a_5, r_5, s_6)	(s_6, a_6, r_6, s_7)	(s_7, a_7, r_7, s_8)
Completed Action	✓	✓	✗	✗
MARL				
Agent 1 Step	-	-	3	-
Agent 2 Step	2	3	-	4
Agent 1 Experience	-	-	(s_3, a_3, r_7, s_7)	-
Agent 2 Experience	(s_4, a_4, r_5, s_5)	(s_5, a_5, r_6, s_6)	-	(s_6, a_6, r_8, s_8)
Completed Action	✓	✓	✓	✓

Table 5.6: State space features of the MARL approach for each agent. The state space cardinality depends on the responsibility of the agent.

Feature	Description
Loader State	Features of associated loader
Position	Current location (==station) of the loader
Gripper Object	Data of loaded object in gripper of the I-loader
Loader State	Features of each neighboring loader
Position	Current location (==station) of the loader
Action	Current executed action
Blocked	Flag when loader can't complete action because other loader blocks its way
Conveyor_In State	Only for the first loader
Available	Flag if a new unprocessed workpiece is available
Station State	Features of each work station that the loader can reach
Occupied	Flag if a workpiece is processed
Finished	Flag if processing has finished
Failed	Flag if the machine has a failure

Table 5.7: Comparison of the rounded state space cardinalities of different approaches for a production line with three work steps and two machines each.
A detailed overview of features for the used states is illustrated in Table A.1.

Approach	State Space Cardinality
SARL	
One I-Loader	3.3×10^5
Two I-Loaders	7.8×10^9
MARL (Responsibility for All Station)	
Agent 1	4.4×10^7
Agent 2	1.8×10^7
MARL (Restricted Work Area)	
Agent 1	9.2×10^5
Agent 2	3.5×10^5

5.3.2 *Action Restrictions*

In an ideal RL scenario, the agent should be allowed to develop its strategy without any predefined restrictions by refining its current approach based on the experiences, which the agent generates by interacting with the environment. It is crucial to acknowledge that the imposition of unsuitable constraints may result in the formulation of an ineffective strategy. Consequently, any restrictions that are introduced should be carefully considered. However, in the context of this project, a few restrictions must be introduced on the agents to guarantee the smooth functioning of the simulation and to prevent actions that could have a significant negative impact on the training process.

The *wait* action is such an action when it is selected in an environment state where no "passive" change can occur anymore. Such a state could occur, for example, at the beginning of an episode when all stations are still empty and an unprocessed workpiece has already arrived at the *Conveyor_In* station. If the agent decides in such a state in favor of the *wait* action, the simulation would remain waiting in this state until the set episode duration has expired. This has the consequence that no more experiences can be observed in this episode, as the remaining simulated time is effectively wasted. To avoid this outcome during the ϵ-greedy state exploration, the "wait" action is temporarily excluded from the random selection in such states. If the agent nevertheless chooses the action based on its current policy, the simulation has a built-in safeguard that cancels the wait after a defined time (see Section 5.1.1).

Further restrictions are imposed to prevent the execution of meaningless actions, such as *loading* when the gripper has already a workpiece loaded or *unloading* when the gripper does not hold a workpiece. It is essential that these actions are prevented until the agents have undergone a sufficient period of training, as they could otherwise cause issues within the used simulation programs. Accordingly, a penalty vector is generated based on the given state. This vector comprises a penalty value for each action that must be restricted; while every allowed action is not penalized. Subsequently, the individual penalty values are subtracted from the output of the DQN, i. e. the approximated Q-values, before the action with the highest Q-value is selected. The structure of the DQN with the penalty vector is illustrated in Figure 5.3.

$$Q(S_t, A_t) = Q(S_t, A_t; \theta) - P(S_t, A_t) \tag{5.1}$$

Equation 5.1 formalizes the calculation of the predicted Q-value with the Q-network with the applied penalty. Similarly, the penalty is integrated into the target value within the Q-learning algorithm. It should be noted that the penalty only influences the selection of the maximum Q-value in the next state, as illustrated in Equation 5.2.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha \left[R_{t+1} + \gamma \max_a \left[Q(S_{t+1}, a; \theta^-) - P(S_{t+1}, a) \right] - Q(S_t, A_t) \right] \tag{5.2}$$

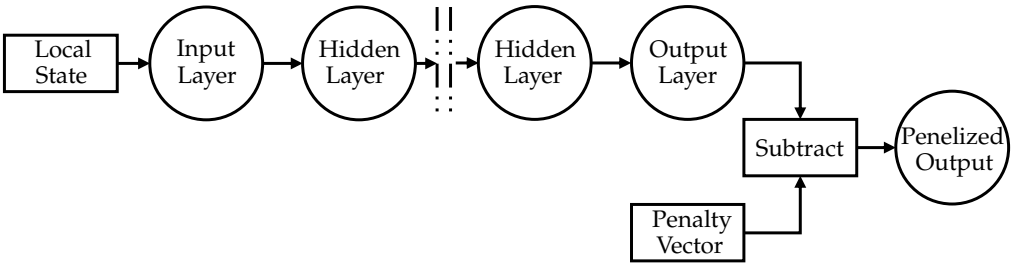

Figure 5.3: Structure of the DQN with the additional penalty vector calculation. The values of the output layer are to be interpreted as approximated Q-values for the individual actions. The Q-values of the restricted actions are reduced through the subtraction of the penalty vector.

The resulting loss function for the weight update of the Q-network is formalized in Equation 5.3

$$L(\theta) = \Big((Q(S_t, A_t; \theta) - P(S_t, A_t)) - \Big[R_{t+1} + \gamma \max_a [Q(S_{t+1}, a; \theta^-) - P(S_{t+1}, a)] \Big] \Big)^2 \tag{5.3}$$

Both the penalty vector P_t for the current State and the penalty vector P_{t+1} for the next state S_{t+1} are included in the experience which is stored in the replay memory.

The utilized restrictions to the action space and the effects of restricting the work areas are explained in Chapter 7.

5.3.3 *Reward Function*

An efficient reward function is an essential element in RL, as it directly impacts the agent's behavior. A well-designed reward function provides guidance to the agents, directing them towards desired actions and ensuring efficient and effective learning. In contrast, inappropriate reward functions may result in unintended behaviors, suboptimal performance, and even exploitation of loopholes in the environment.

As a first approach, a fairly simple reward function is designed:

- Unloading a finished workpiece at the *Conveyor_Out* station is rewarded with the value 5
- Unloading a workpiece from the gripper into the correct work station is rewarded with the value 2.
- Loading a workpiece into the gripper of a loader is rewarded with the value 1.
- If a loader waits, while it is blocking another loader, this waiting is penalized with the reward value -3.
- All other actions, such as a drive to another station or waiting in general, are not rewarded.

The SARL approach employs a single DQN, which consequently utilizes a single reward function for both loaders. Therefore, a suboptimal strategy may be learned where the

agent attempts to send the first loader in the vicinity of the *Conveyor_Out* station, as a higher reward can be expected near the output, despite the inability to drive towards it. In contrast, the MARL approach divides the system into two agents, each with its own independent reward maximization problem. Although both reward functions may be based on the same set of rules, the splitting ensures that the potential for the second agent to achieve a higher absolute reward has no impact on the training of the network from the first agent.

Upon further examination, it becomes evident that the absence of the reward for delivering a finished workpiece still may affects the strategy of the first agent. The unloading of a workpiece at *Station_A* and *Station_B* is equally valuable in terms of the achieved reward as unloading at *Station_C* and *Station_D*.

We assume the following scenario: A new workpiece is available at the *Conveyor_In* station, *Station_B* has finished the processing of the loaded workpiece, and the loader is currently located at *Station_A*, which is not occupied. In this context, the agent has two equally rewarded action sequences (excluding execution times):

- Filling *Station_A* with the unprocessed workpiece from *Conveyor_In*:
  Conveyor_In (0) $\rightarrow$ *load* (1) $\rightarrow$ *Station_A* (0) $\rightarrow$ *unload* (2)
- Transporting the workpiece from *Station_B* to *Station_C* or *Station_D* (assuming these stations are currently not occupied):
  Station_B (0) $\rightarrow$ *load* (1) $\rightarrow$ *Station_C* (0) $\rightarrow$ *unload* (2)

It can be observed that the reward sequence and the total reward are identical for both potential pathways. However, when considering the overall production line, it would likely be more optimal to prioritize the transfer of the workpiece from *Station_B* to a station of the subsequent work center, as this allows the second loader to receive new input for its designated work area at an earlier stage.

One potential solution is the use of time-discounted future rewards, which reduce the value of future rewards depending on how long the execution of the selected action takes. This is formalized in Equation 5.4, with d_{t+1} as the difference in the simulation time between step t and $t+1$.

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma^{d_{t+1}} \max_a Q(S_{t+1}, a) - Q(S_t, A_t)] \tag{5.4}$$

Furthermore, the duration is also stored in the replay memory, as it cannot be calculated at a later point in time.

In practice, however, there is no guarantee that the travel times between stations will be different. Therefore, an extension of the reward function is considered so that unloading a workpiece no longer results in a constant value, but the reward is awarded depending on the work step. Unloading at work center one (*Station_A* and *Station_B*) will result in a reward value of 1; unloading at work center two (*Station_C* and *Station_D*) will result in a reward of 2, and so on. In addition, the reward for delivering a finished workpiece at the *Conveyor_Out* station will also be adjusted, so that this reward is one greater than unloading at the last work center. This graduated reward is designed to prioritize the quick transfer of the workpieces through the production line.

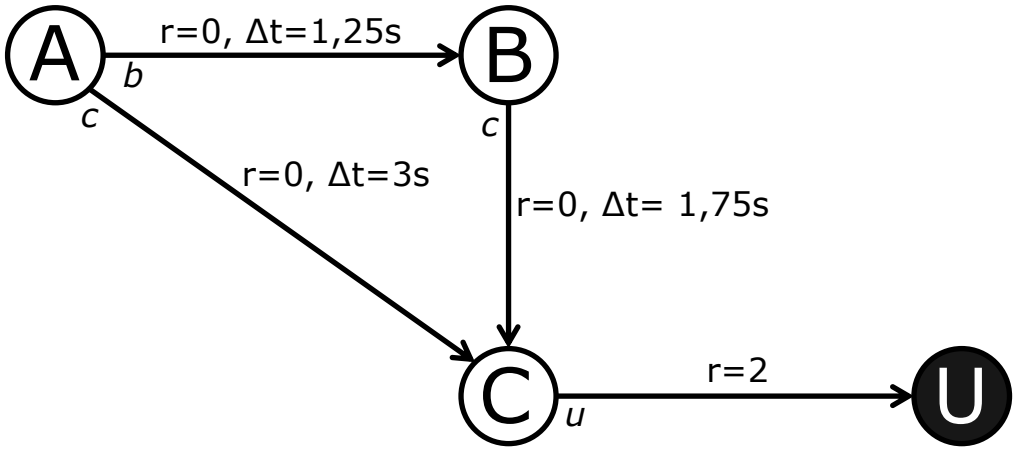

Figure 5.4: Sketched Markov Decision Process with reward and action duration.
The normal circles represent the loader position at the specified station. The filled circle **U** represents the finite state after unloading a workpiece at *Station_C*. The lowercase letters symbolize the required action to transition to the subsequent state.

The simulation uses a simplification when calculating the execution times of drives between stations. Here, the travel times follow discrete values instead of taking the acceleration and deceleration of the loaders into account. As a result, a consecutive drive from *Station_A* to *Station_B* and then from *Station_B* to *Station_C* will take the same amount of time as a direct drive from *Station_A* directly to *Station_C*. In practice, the direct drive would be faster, since there would be not time lost, due to unnecessary stops.

Figure 5.4 illustrates this problem in a very simplified way as a decision process with only four states. The states **A**, **B**, and **C** represent the location of the loader at the named station. The final state **U**, occurs after the initial loaded workpiece has been unloaded at *Station_C* The lowercase letters symbolize the required action to transition to the subsequent state. The action to transition to the next state is defined as the lowercase of the state name, e. g. b, c, u.

In accordance with the theory of RL, the agent should, over time, learn to select the direct path ($\mathbf{A} \rightarrowtail \mathbf{C} \rightarrowtail \mathbf{U}$). This is due to the fact that the reward for unloading is only one step away, in contrast to the two-step path involving an intermediate stop.

Equation 5.5 presents the calculated values for the Q-values in state **A** for both paths. The calculation is based on Equation 4.7 with a discount factor $\gamma = 0.99$ for the Q-values of the subsequent state.

$$\begin{aligned} q(\mathbf{A}, b) &= 0 + 0.99 * (0 + 0.99 * (2)) &&= 1.9602 \\ q(\mathbf{A}, c) &= 0 + 0.99 * (2) &&= 1.98 \end{aligned} \tag{5.5}$$

It is evident that the path with an intermediate stop results in a lower Q-value. However, in order to make more optimal decisions in other states, it is reasonable to include the execution time of the actions in the calculation of the future return (see Equation 5.4). As anticipated, the Q-values calculated with the time-discounted future return are slightly

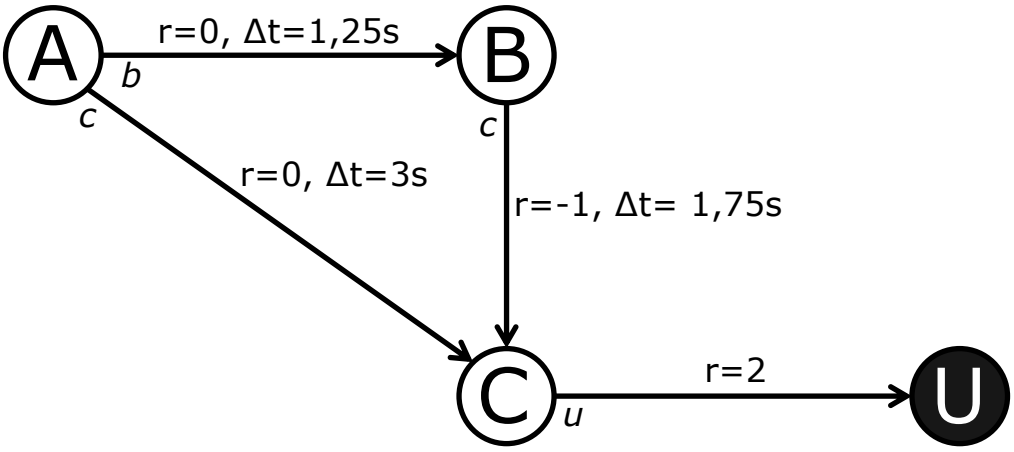

Figure 5.5: Sketched Markov Decision Process with reward (incl. applied penalty for consecutive drives) and action duration.
The normal circles represent the loader position at the specified station. The filled circle **U** represents the finite state after unloading a workpiece at *Station_C*. The lowercase letters symbolize the required action to transition to the subsequent state.

lower (see Equation 5.6). However, the discrepancy between the two paths has also diminished.

$$\begin{aligned} q(\mathbf{A}, b) &= 0 + 0.99^{1.25} * (0 + 0.99^{1.75} * (2)) && \approx 1.9406 \\ q(\mathbf{A}, c) &= 0 + 0.99^{3} * (2) && \approx 1.9565 \end{aligned} \tag{5.6}$$

In consideration of the entire system, it is uncertain whether such a minor discrepancy can be effectively learned, given that the Q-value for the actual system depends upon a multitude of additional factors and is approximated by the DQN. For these reasons, a negative reward is introduced that penalizes consecutive drives. Figure 5.5 illustrates the same minimal example where the second drive action from state **B** to state **C** is now penalized with the reward -1.

Equation 5.7 shows the Q-values for the longer path for both with and without time-discounted future rewards. These values are now, as expected, significantly smaller in comparison to the Q-values for the direct path.

$$\begin{aligned} q(\mathbf{A}, b) &= 0 + 0.99 * (-1 + 0.99 * (2)) && = 0.9702 \\ q(\mathbf{A}, b) &= 0 + 0.99^{1.25} * (-1 + 0.99^{1.75} * (2)) && \approx 0.9531 \\ q(\mathbf{A}, c) &= 0 + 0.99 * (2) && = 1.98 \\ q(\mathbf{A}, c) &= 0 + 0.99^{3} * (2) && \approx 1.9565 \end{aligned} \tag{5.7}$$

However, the introduction of a penalizing reward for consecutive driving also presents a potential disadvantage. As illustrated in Figure 5.6 the state **B** may also be reached from state **X** through the execution of action x, which was not a driving action. Therefore, no penalty would be incurred for the subsequent drive from state **B** to state **C**. Consequently, the state-action-state transition (**B**, c, **C**) is associated with two distinct reward values (in this example -1 and 0). The reward which is awarded for the drive from **B** to **C** depends now not only on the current state s_t (**B**) but also on the previous state s_{t-1} (**A** or

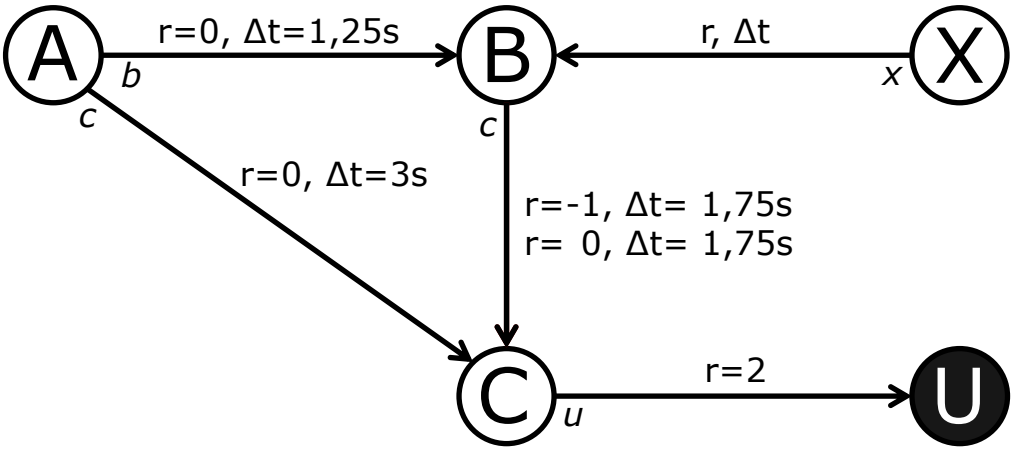

Figure 5.6: Sketched Markov Decision Process with inconsistent rewards and action duration. The normal circles represent the loader position at the specified station. The filled circle **U** represents the finite state after unloading a workpiece at *Station_C*. The lowercase letters symbolize the required action to transition to the subsequent state. State **X** is an undefined state, which transition x to state **B** is not a drive action.

X) or rather the previous action a_{t-1} (b or x). This stands in contrast to the fundamental tenet of a Markov Decision Process (MDP), which posits that the subsequent state s_{t+1} and the reward r_{t+1} only depends on the current state s_t and the chosen action a_t (see Section 4.2).

In order to maintain consistency with the MDP principle, an additional flag feature could be incorporated into the state, indicating whether the preceding action was a driving action. This would effectively double the state space, but it would not be feasible to integrate this flag into the used simulation software. An alternative, more straightforward approach would be to penalize all driving actions.

The effects of these reward function adjustments on the training are shown and explained in Chapter 7.

6 IMPLEMENTATION

This chapter presents a summary of the implementation for the Multi-Agent Reinforcement Learning (MARL) approach described in Chapter 5. The implementation builds upon the existing structure and the interface for the simulation software from the project KI-Verfahren zur Steuerung von Digitalen Portalroboterzwillingen (KISPo) [Zis+24]. The primary programming language utilized for this project is Python, with TensorFlow [Aba+15] employed as the deep learning framework. Furthermore, the diagrams and figures included in this chapter were generated using PlantUML [Pla]. In the second section the implementation of the proposed MARL methodology is presented, illustrating how the theoretical concepts are translated into a functional system.

6.1 COMMUNICATION WITH THE SIMULATION

The production line is modeled through the industry-standard software Plant Simulation and AnyLogic. For the communication between the simulation and the learning framework, an interface was established within the KISPo project. The interface utilizes a local web server embedded in the implemented learning framework to which the simulation transmits Hypertext Transfer Protocol (HTTP) requests.

Figure 6.1 illustrates the communication process between the simulation and the learning framework. The configuration of the production line is initially requested from the simulation. This procedure must be carried out in the initial stage of each episode, due to the intrinsic functionality of the simulation programs, despite the fact that the configuration remains constant for the whole training process. Afterward, the simulation notifies the learning framework so that it is prepared to start. At this stage, the learning framework may overwrite the initial state of the simulation.

The following inner loop of the procedure is the actual simulation and interaction process. Therefore, the simulation transmits its current state to the learning framework, which determines the action based on the selected algorithm. Subsequently, this action is executed within the simulation, resulting in the transmission of the next request, which contains the new simulation state and the assigned reward. This sub process is repeated until the defined simulation time is reached. This termination of the episode is signaled to the algorithm via the sim-end route. The episode is also terminated when an error or a critically incorrect action has been selected. Finally, the response to the sim-end route initiates the next episode in the simulation. This process repeats until the desired number of episodes is reached.

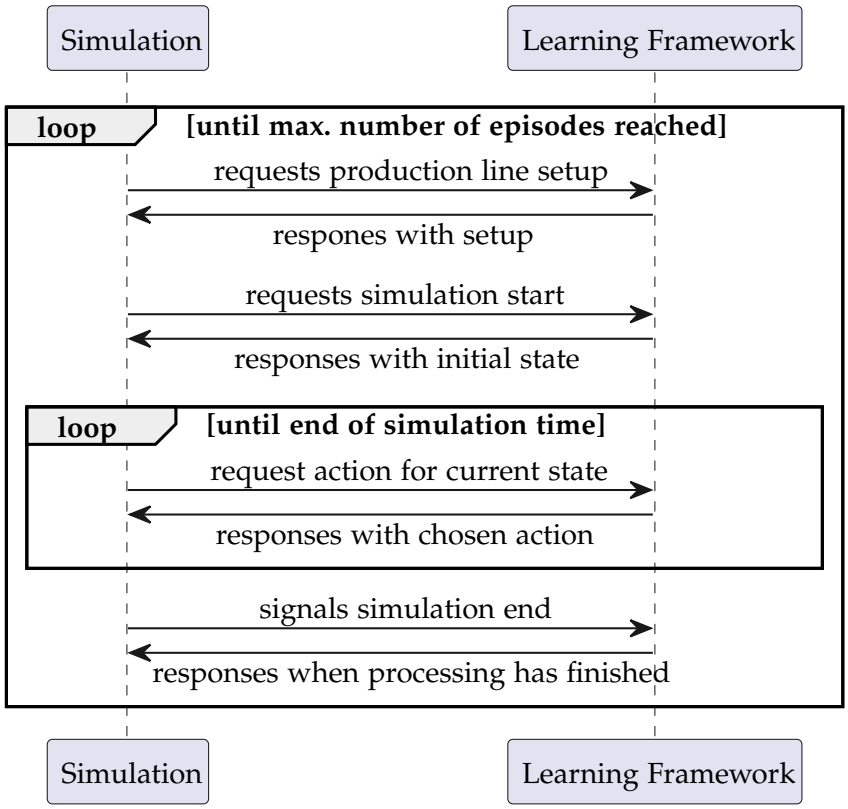

Figure 6.1: Sequence diagram of the communication between simulation and the learning algorithm.

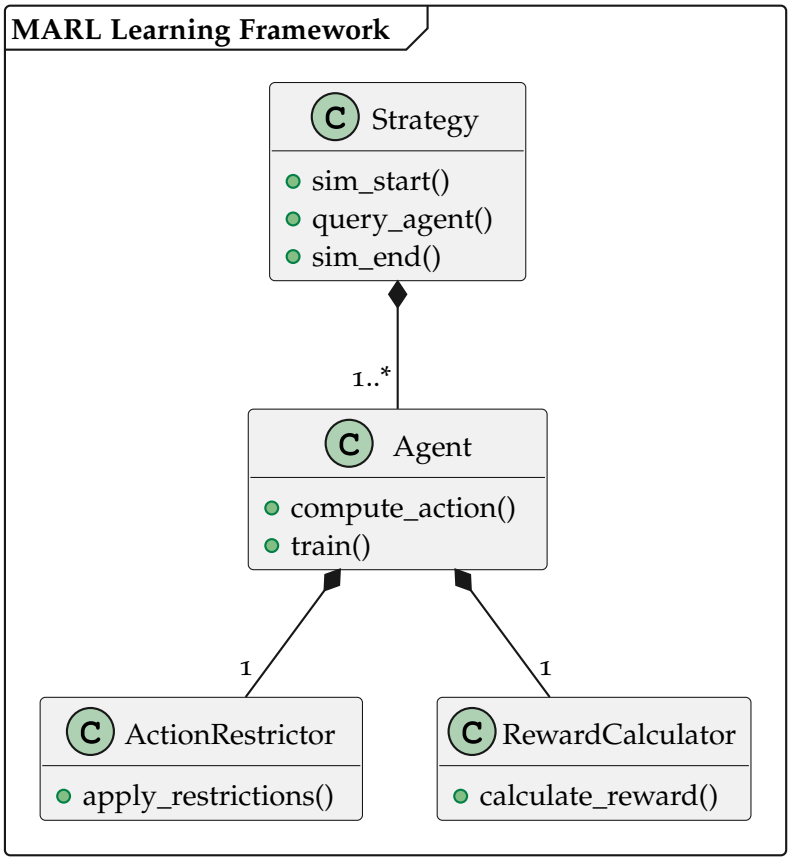

Figure 6.2: Simple class diagram of the MARL learning framework with its components.

6.2 MARL IMPLEMENTATION

Figure 6.2 provides a simplified class diagram for the implementation of the MARL approach. The simplification is limited to the necessary components and their main methods for the learning procedure; all other classes, e.g., for the communication interface, are not shown for clarity. A detailed class diagram is illustrated in the appendix in Figure B.1.

The following subsections explain the structure and tasks of the four components.

6.2.1 *Training Strategy*

The strategy class serves as the primary control unit for all processes within the learning framework. Requests from the simulation are routed to the designated method (*sim_start*, *query_agent*, *sim_end*) via the local web server.

The processing of the setup request and the return of the production line configuration are conducted outside of the actual MARL implementation, as this response remains constant for the entire training run. The initial state of the simulation must be overwritten during handling the *sim_start* request, since the simulation positions the loaders in order by default, with the first loader at the *Conveyor_In* station and the second loader at *Station_A*. However, the restriction of the working area (see Section 5.3.1) prohibits the second loader from occupying this position, so the initial position is set to *Station_C*.

Upon the simulation's request for a new action, the requesting loader is initially extracted from the query. With the assistance of an assignment table, constructed during the initialization phase, the query is passed to the designated agent instance, which is responsible for the requesting loader. Subsequently, the training of the agents' Deep Q-Network (DQN) is initiated with an adjustable probability (in this case, 5 %). The motivation behind the decision to train during the episode, as opposed to only at its conclusion, is twofold. Primarily, this approach is intended to reduce the overall number of episodes required. Secondly, the design aims to prevent the agents from becoming "stuck" within an episode. The term "getting stuck" is used to describe a situation in which the agent's actions result in a lack of progress within the episode, due to inadequate training outcomes. One potential scenario, is that the agent will consistently select the *wait* action, as this receives the highest Q-value approximation for the current state. Training during the episode may result in a slight adjustment to the weights of the DQN, leading to the selection of an alternative action for the same state. This is not a prerequisite as long as the ϵ-greedy value remains relatively high at the beginning of training, as the random actions will also result in a transition to another state. However, as training progresses, the ϵ value is reduced (to 0.1 %), resulting in the infrequent occurrence of random actions.

Upon the completion of the episode, the simulation transmits its final state. However, this state is not utilized, as it represents an "intermediate" state in which the previous action has not been fully executed. Instead, the agent's processing of the experience gained during the episode is initiated. Afterward, the agents' DQN are once again trained, and the update of the weights of the target network is initiated every ten episodes.

Furthermore, the decay of the ϵ-greedy approach is controlled from the strategy, as the ϵ-value is the same for all agents. The training starts with an initial ϵ-greedy value $\epsilon_0 = 1$. The reduction is dependent upon the used settings and occurs either upon the beginning of the network training, contingent upon a required number of entries in the replay memory, or when the replay memories of the agents are fully populated (in this case, 10^5 entries). The decay itself follows a power function, whereby the ϵ-value is multiplied each episode by a defined decay rate λ_ϵ:

$$\epsilon_{new} = \epsilon * \lambda_\epsilon \quad \text{with} \quad \lambda_\epsilon \in (0,1) \tag{6.1}$$

The ϵ-value is reduced until the minimal value $\epsilon_{min} = 0.001$ is reached. As the final step necessary statistics for the episode are recorded and all components are prepared for the next episode.

6.2.2 *Deep Q-Network-Agent*

During the initialization of the learning framework, a unique agent instance is created for each loader. This instance receives all necessary information, including details about the loader itself, neighboring loaders, and accessible work stations. Based on this information, the agent's local action space A_i is created and the number of input and output nodes of the DQN is calculated in order to create both the main NN and the target network. Each agent also creates an instance of the *ActionRestrictor* class and the *RewardCalculator* class. Each agent is equipped with two distinct memory systems for the purpose of storing observed experiences. The replay memory stores the experiences, which are utilized in the generation of the training data for the DQN training. Each entry is a data tuple, expressed as $(s_t, p_t, a_t, r_{t+1}, s_{t+1}, p_{t+1}, d_{t+1})$, where s_t is the current local state and p_t is the corresponding penalty vector, and a_t is the chosen action. Furthermore, the tuple includes the achieved reward r_{t+1}, the next state $s_{t+1}$and its corresponding penalty vector p_{t+1}, and the duration d_{t+1} of the execution of the action.

As indicated by the index values, the data originate from disparate time steps, specifically t and $t+1$. It is therefore essential to utilize a supplementary memory, namely the episode memory, which serves as a storage for data accumulated over the course of an episode, retaining information from a single time step. At the end of an episode, the entries of the episode memory are pairwise linked to form a complete experience, which is then stored in the replay memory.

Both memories are implemented as ring buffers with a predefined maximal length (in this case, 10^5 entries). Once the ring buffer is at maximum capacity, the addition of a new entry results in the removal of the oldest entry. The PYTHON *deque* object is employed for this purpose. A *deque* is a special type of list where the addition and removal of objects are approximately $O(1)$ time complex, as opposed to the $O(n)$ time complexity associated with a normal list [Pyt].

Upon the simulation's request for an action, the strategy class passes the simulation state to the agent instance that is responsible for the requesting loader. As the preliminary step in the selection of the action, the simulation state is passed to the *ActionRestrictor*

instance. Based on the configured restriction, the list of permitted actions and the penalty vector is created for the current state. Moreover, the agent generates the local state from the query in accordance with its designated responsibilities. During this generation, the values of the selected features of the simulation state, which is formatted as JavaScript Object Notation (JSON), are mapped to numbers, as the local state is used as input data for its DQN. In the second step, a probability of the current ϵ-value is used to determine whether a random action is selected or whether the action with the highest approximated Q-value is selected. In the event that the ϵ-greedy approach is selected, a random action is chosen from the list, returned by the *ActionRestrictor* beforehand. In the alternative scenario, the local state and the penalty vector are employed to predict the Q-values with the DQN.

In the subsequent phase, the reward associated with the previous action is passed to the *RewardCalculator* instance, which may modify the reward in accordance with the configured reward function adjustments. Finally the tuple $(s_t, p_t, a_t, r_t, d_t)$ for the current time step t is stored in the episode memory. The tuple consists of the current local state s_t, the current penalty vector p_t, the selected action a_t, along with the reward r_t and the execution duration d_t of the previous step.

At the completion of each episode, the entries from the episode memory are processed in pairs. During this processing, the information of each pair $(s_t, p_t, a_t, r_t, d_t)$ and $(s_{t+1}, p_{t+1}, a_{t+1}, r_{t+1}, d_{t+1})$ for $t = 0, 1, .., T-1$ with $T = |\textit{Episode Memory}|$ is combined to the experience $(s_t, p_t, a_t, r_{t+1}, s_{t+1}, p_{t+1}, d_{t+1})$, which is stored in the replay memory.

The training of the DQN may be initiated either during the episode or following the episode's completion. In both scenarios, the training data must be derived from the experiences stored in the replay memory. Listing 6.1 demonstrates the generation process in a simplified function. First, a random sample of experiences is selected from the replay memory. Secondly, the input data (*x_train*) is created by extracting the current state and current penalty from each experience. To create the target data (*y_train*), the Q-values for the current state and the subsequent state are predicted. Subsequently, the current Q-value of the selected action of each experience is overwritten with a new target Q-value, which is calculated using the adjusted Q-learning algorithm (see Equation 5.4). The list of updated Q-values then resembles the target data. Finally, the data pair is passed to the TENSORFLOW framework, which performs the weight update of the DQN.

6.2.3 *Restrictions*

Each agent initializes its own *ActionRestrictor* class instance and transmits its responsibility configuration, which includes the reachable stations and the neighboring loaders. This is a requisite step, as the restrictions are only evaluated on the agent's local state space S_i and subsequently applied to the local action space A_i. Upon the simulation's request for a new action, the agent transmits the global state to its *ActionRestrictor* instance. During the processing, first, all actions that should be restricted are temporarily removed from the agent's local action space. To illustrate, in a state where the gripper of the requesting loader already holds a workpiece, the *load* action is removed from

```python
def generate_training_data(replay_memory, batch_size, main_model, target_model):
        # 1: select a random batch from the replay memory
        experiences = sample(replay_memory, batch_size)
        # 2: create x_train
        current_states = [exp.state for exp in experience]
        current_penalty = [exp.penalty for exp in experience]
        x_train = [current_states, current_penalty]
        # 3: create y_train
        next_states = [exp.next_state for exp in experience]
        next_penalty = [exp.next_penalty for exp in experience]
        # 3.1: get Q-value approximations
        current_q_values = main_model.predict(x_train)
        next_q_values = target_model.predict([next_states, next_penalty])
        # 3.2: update target Q-value of selected action
        for i, exp in enumerate(experiences):
                # calculate new target Q-value
                target_q = exp.reward + discount_factor**exp.duration * max(
                    next_q_values[i])
                # overwrite the current Q-value of the action
                current_q_values[i][exp.action] = target_q

        y_train = current_q_values
        return x_train, y_train
```

Listing 6.1: DQN Training Data Generation from the experiences.

the local action space A_i. This subset of the local action space is used for the ϵ-greedy approach. However, if the action is selected by the DQN the penalty vector p is needed (see Section 5.3.2). The penalty vector maps the complete local action space A_i and assigns a penalty value to each disallowed action. Both, the list with allowed action and the penalty vector, are returned to the agent.

6.2.4 *Reward Function*

In contrast to a direct reward value, the simulation only transmits the reward type (e.g., *CorrectLoad*, *CorrectStep*, *Finished*). This enables a more flexible configuration of the actual reward value within the learning framework. Consequently, modifications to the reward function can be implemented with greater flexibility.

The responsibility for performing this task is assigned to the *RewardCalculator* class. The initial step is to convert the type of reward received into the configured value. In the case of the graduated reward, an increasing value is assigned in contrast to the fixed values assigned to the *CorrectLoad* and the *Finished* reward types. This implies that unloading at the first work center results in a reward of 1, unloading at the second work center results in a reward of 2, and so on.

Subsequently, it is verified whether consecutive drives have been recorded. Therefore, the *RewardCalculator* receives the last two executed actions. In the event, that both are driving actions, the existing reward value is reduced by the corresponding penalty value. Finally, the remaining reward value is returned to the agent.

7

EXPERIMENTS AND RESULTS

This chapter examines the potential of the proposed approach for controlling multiple gantry robots, as outlined in Chapter 5, by conducting a series of experiments. The objective of the individual experiments is to demonstrate the suitability of the Multi-Agent Reinforcement Learning (MARL) approach in the context of the production plant and the impact of the various modifications on the training of the Deep Q-Network (DQN). Initially, the default configurations for the experiments and the used criteria to evaluate the quality of the training models are described. Afterward, the results of the Single-Agent Reinforcement Learning (SARL) approach are presented as a reference for subsequent analyses of the different MARL strategies. Then six different experiments are conducted in a step-by-step manner, with each subsequent experiment building on the previous one. This enables the evaluation of the enhancements outlined in the methodology (see Chapter 5). At the beginning of each experiment, the modifications to the experimental configuration are described. The analysis and evaluation of the results are conducted in accordance with the established evaluation criteria.

7.1 EXPERIMENT SETUP

The following experiments were performed on the same production line layout, as illustrated in Figure 7.1. Each workpiece must undergo three different work steps before it can be considered complete. Two machines are available for each work step, with processing occurring at a constant rate of ten seconds. Additionally, new unprocessed workpieces are delivered to the *Conveyor_In* at a rate of one per ten seconds. Two gantry robots of the "I" type are employed for the transportation of workpieces between stations. Each I-loader is equipped with a single gripper. The execution duration of the different actions is illustrated in Table 7.1. In the absence of obstructions, the travel

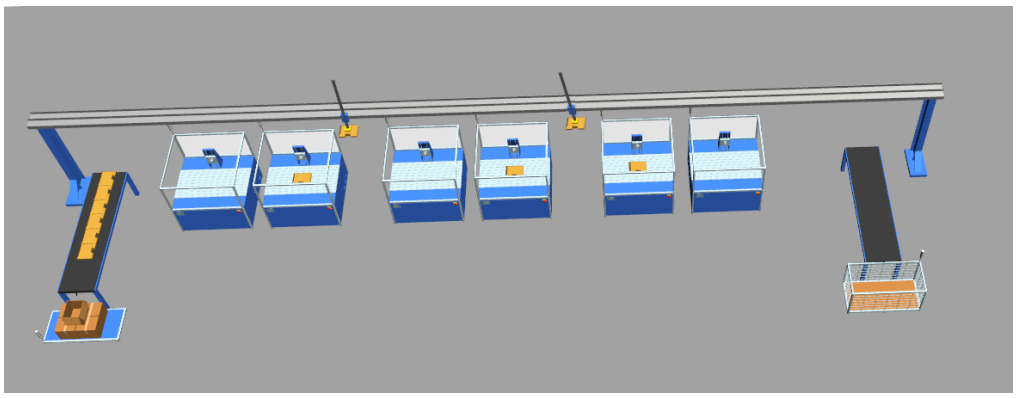

Figure 7.1: Digital twin of the production line used in the experiment. Two I-loaders and three work steps with each two machines are employed.

Table 7.1: Execution duration of the distinct actions. The stations are located at varying distances from one another. In the case of drives over multiple stations, the individual travel times are summed up.

Action	Execution Duration
load	2.8 s
unload	2.8 s
Conveyor_In $\Longleftrightarrow$ *Station_A*	1.5 s
Station_A $\Longleftrightarrow$ *Station_B* *Station_C* $\Longleftrightarrow$ *Station_D* *Station_E* $\Longleftrightarrow$ *Station_F*	1.25 s
Station_B $\Longleftrightarrow$ *Station_C* *Station_D* $\Longleftrightarrow$ *Station_E*	1.75 s
Station_F $\Longleftrightarrow$ *Conveyor_Out*	2 s

times between stations exhibit consistent values. In the event of a drive traversing multiple stations, the duration of that drive is the sum of the individual drives. To illustrate, the duration of the drive from *Conveyor_In* to *Station_D* is calculated with $d = 1.5\,\text{s} + 1.25\,\text{s} + 1.75\,\text{s} + 1.25\,\text{s} = 5.75\,\text{s}$. To clarify the impact of varying methodologies, machine failures are not included in the experiments, with the exception of the one presented in Section 7.7.

The agents undergo training over the course of several episodes, with each episode covering 20 minutes of simulated time. The standard training period for MARL experiments consists of 2500 episodes. At the beginning of each episode, all work stations are unoccupied, indicating that there are no workpieces present within the production line. The utilized epsilon decay rates λ_ϵ for the ϵ-greedy approach are dependent upon the complexity of the state space. The chosen decay rates and their respective progressions are illustrated in Figure 7.2. The epsilon value, which regulates the frequency of the ϵ-greedy approach, is decreased with each episode. This process continues until the specified minimum value $\epsilon_{min} = 0.001$ is reached. The decay rates are set in such a way that the minimum is reached after a certain number of episodes that are indicated by the vertical lines in Figure 7.2. It is important to note that the illustration only presents the theoretical progression of the decay. In practice, the decay will only begin once the DQN has also commenced its training. In addition to training at the completion of each episode, training also takes place with a probability of 5 % during the episode following the processing of the request for a new action. On average, training is performed with this set probability approximately 45 times per episode.

Each agent, whether utilizing the SARL or MARL approach, operates with a replay memory capacity of 10^5 entries. Once the replay memory reaches a capacity of 10^4 entries, the training of the DQN and the epsilon decay is initialed. This delay is intended to ensure the availability of data exhibiting sufficient variance for training of the DQN. The training itself is conducted using a single batch comprising 64 random experiences from the replay memory. The structure of the DQN is consistent across all experiments,

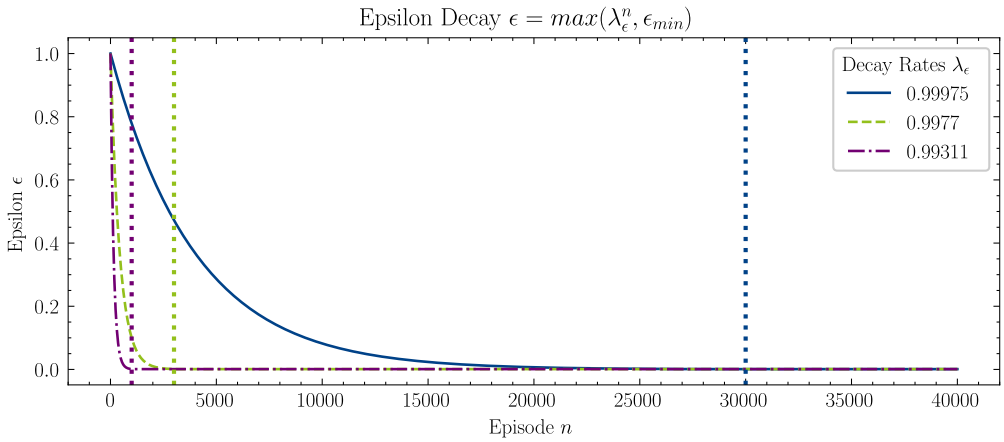

Figure 7.2: Progression of the ϵ value over the training. The vertical lines indicate the episode when the decay has reached the minimum epsilon value $\epsilon_{min} = 0.001$.

with the exception of the number of neurons in the input and output layer, which varies based on the utilized state and the size of the action space of the corresponding agent. Each DQN comprises three hidden layers, each with 32 neurons, which are activated via the Rectified Linear Unit (ReLu) function. In the case of the output layer, which represents the approximated Q-values, the linear activation function is employed, whereby the values are output without undergoing any further modification. With regard to the Q-learning algorithm (see Section 4.2.3), a discount factor of $\gamma = 0.99$ is utilized for future rewards. It should be noted that time discounting of the future reward is only applicable in the context of MARL approach, given that the execution duration obtained from the simulation cannot be employed due to the processing of queries in the SARL approach (see Section 5.2).

In accordance with the default reward function (Equation 7.1), the action a_t is assigned a reward R_{t+1} in accordance with the following rules:

$$R_{t+1} = \begin{cases} 5, & \text{unloading a finished workpiece at } \textit{Conveyor\_Out} \\ 2, & \text{unloading a workpiece at a work station} \\ 1, & \text{loading a workpiece from } \textit{Conveyor\_In} \text{ or a work station} \\ -3, & \text{waiting at a station while blocking other loader} \\ 0, & \text{otherwise} \end{cases} \tag{7.1}$$

The following restrictions are employed by default. If a restriction is triggered in the current state, the corresponding action is removed from the list of allowed actions for the ϵ-greedy approach and penalized in the penalty vector, when the action is selected by the DQN (see Section 5.3.2).

1. *Loading* when the gripper has already a workpiece loaded
2. *Loading* from an empty station
3. *Loading* at a work station that has not finished its process

4. *Loading* at the *Conveyor_Out* station
5. *Loading* a workpiece when all stations of the following process step are occupied
6. *Unloading* when the gripper has no workpiece loaded
7. *Unloading* a workpiece at the *Conveyor_In* station
8. *Unloading* a workpiece at an occupied work station
9. *Unloading* a workpiece at a work station that performs the wrong processing step
10. *Driving* to the station where the requesting loader is already located
11. *Driving* to stations that are behind the neighboring loader

An example of the fifth restriction can be observed in the scenario where *Station_A* and *Station_B* are already processing a workpiece. In such a scenario, the agent is precluded from loading a new workpiece from the *Conveyor_In* station, as it cannot be unloaded as long as both stations remain occupied. If multiple loaders can reach those stations, this restriction may be overly strict; otherwise, the result would be a deadlock with no further productive actions in the running episode. The ninth restriction is intended to ensure compliance with the prescribed sequence of process steps. The eleventh restriction is only triggered when multiple loaders are employed. For example, if the second loader is located at *Station_C*, the first loader is prohibited from driving to *Station_D*, *Station_E*, and *Station_F*.

7.1.1 *Evaluation Criteria*

In theoretical terms, the primary criterion for an effective strategy is to achieve the highest possible hourly throughput. The maximal achievable throughput depends on a number of factors, including the number of work steps required and the processing time on the machine. Furthermore, an effective strategy should demonstrate superior performance in comparison to straightforward heuristics or, as utilized in these experiments, the First In, First Out (FIFO) strategy, which serves as a fundamental point of reference. The FIFO strategy is implemented as a basic job queue. Once a workpiece has been processed at a work station, a job is created in the queue for further transport to the next station. The loaders select the oldest job from the list and execute it.

During the training process of the Reinforcement Learning (RL) strategies, a straightforward metric is utilized to determine the so-called "best" model. This metric calculates the moving average of the achieved throughput over the previous ten training episodes. In the event that this moving average is higher in a given episode than the previously determined highest average, the corresponding model persists as the "best" model.

In practice, however, it is also beneficial to achieve the highest possible throughput with the fewest possible actions and movements, as these exert unnecessary strain on the mechanical components. Moreover, suboptimal actions, such as consecutive drives rather than a direct drive to a distant station, also increase the necessary communication

between the simulation and the agent. The experiments conducted during the course of this thesis have demonstrated that the highest throughput is achieved at an early stage of the training with the MARL approach, although a considerable number of suboptimal actions still occur. Consequently, the evaluation is not limited to the "best" model but also encompasses the model that is ultimately generated at the conclusion of all training episodes. In the following sections, this model will be referred to as the "final" model.

To ensure efficient testing of various production system configurations, it is advisable to minimize the duration of the training process. This can be achieved by limiting the number of episodes and ensuring that each episode is as brief as possible. Furthermore, these criteria must be met in scenarios where simulated machine failures are present. This guarantees that the highest throughput is achieved even in instances of temporary "production bottlenecks".

The "best" and "final" models derived from each training run are validated through the simulation of 100 episodes. The validation process is conducted without any actions selected by the ϵ-greedy approach. The performance indicators measured during the validation are the hourly throughput and the number of actions required to reach that throughput. As each validation episode begins with an empty production line, a settling phase with reduced achieved throughput is observed until the main strategy is executed. To mitigate the impact of this settling phase on the metrics, each episode is simulated for a duration of two hours.

7.2 REFERENCE EXPERIMENT: SINGLE-AGENT

This section presents the results of two reference training runs with the SARL implementation, in which one or two I-loaders are utilized within the production line. To guarantee the success of the training process with two I-loaders, it is essential to reduce the ϵ-decay to provide adequate exploration of the large global state space. The utilized epsilon decay $\lambda_\epsilon = 0.999\,77$ results in the minimum ϵ-value $\epsilon_{min} = 0.001$ being reached after around 30 000 episodes. This is necessary since the cardinality of the state space has increased to approximately 7.8×10^9 distinct states when the second loader is introduced (see Section 5.2.4). Additionally, the number of training episodes is significantly increased to 40 000 to compensate for the slower epsilon decay and to train the DQN more frequently.

Figure 7.3 illustrates the training progress based on the hourly throughput achieved. For purposes of comparison, the training progress of a model where only a single I-loader is employed is also delineated in the figure. As anticipated, it can be observed that the utilization of a second loader results in an increased throughput due to the potential for parallelization in the further transportation of workpieces. However, it is also evident that the training runs exhibit considerably higher variance, with numerous instances where only a low throughput is achieved. Even after the epsilon value has reached its minimum, there are still numerous episodes with low throughput.

Table 7.2 presents the results of the validation of both "best" models. A comparison of the throughputs achieved between the training and validation phases indicates that a slightly higher throughput is achieved during the validation phase. This discrepancy

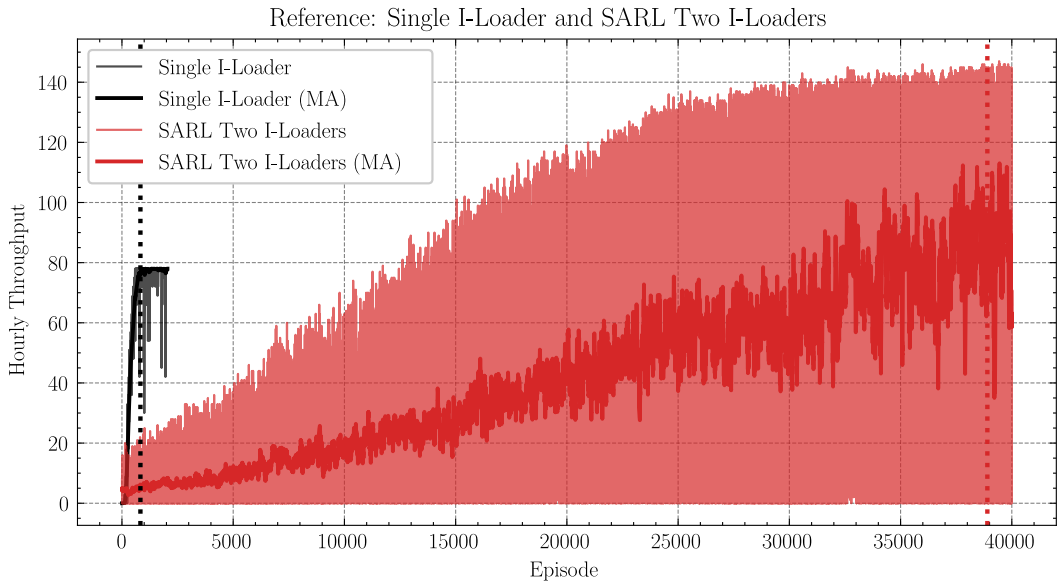

Figure 7.3: Hourly throughput of training runs used as a reference for the new MARL approach. To enhance visibility, a moving average is calculated over 50 episodes.

Table 7.2: Results of the validation of the "best" model for the reference strategies. All values are presented on an hourly basis. The column labeled "FIFO Throughput" indicates the achieved throughput when the FIFO strategy is utilized.

Strategy	Throughput	Actions	FIFO Throughput
Single I-Loader	82	1360	74.5
SARL Two I-Loaders	153	2600	131

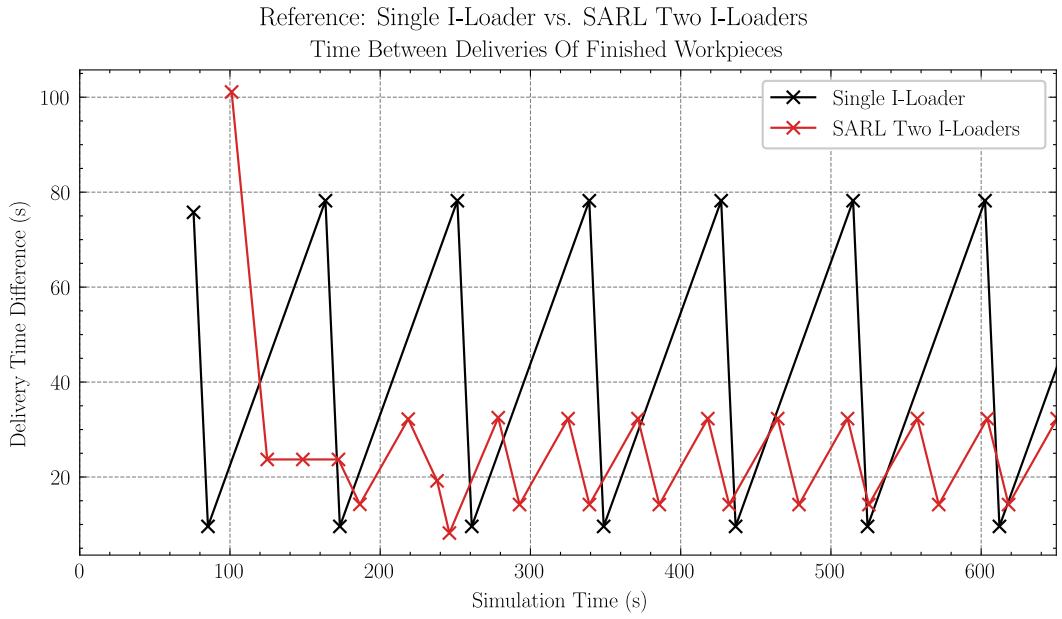

Figure 7.4: Comparison of the time between delivery of finished workpieces at the *Conveyor_Out* station for both reference strategies.

can be attributed to two primary factors: Firstly, the use of the ϵ-greedy approach during the training may results in suboptimal actions that disrupt the learned strategy and therefore result in a decrease in the throughput. Secondly, the simulation of two hours during the validation reduces the impact of the settling phase of the system on the overall throughput value. A notable observation in the comparison of required actions is that the number nearly doubles when two loaders are employed, yet the throughput only increases by approximately 80 %. Furthermore, it is evident that both learned strategies outperform the FIFO strategy.

Figure 7.4 illustrates the interval between the delivery of finished workpieces at the *Conveyor_Out* station during the initial 600 seconds of a randomly selected validation episode. A settling phase of approximately 250 seconds can be observed for the SARL strategy, which utilizes two I-loaders. During this time interval, the duration between deliveries occurs in an irregular pattern. Following the settling phase, the pattern remains consistent until the completion of the simulation. In contrast, the single I-loader strategy results in the delivery of all workpieces, with the exception of the initial one, at a regular rhythm.

7.3 EXPERIMENT 1: MULTI-AGENT

In this initial experiment, the MARL approach proposed in this thesis is evaluated in its most basic form (see Section 5.3). This involves contrasting with the SARL approach from the reference experiment, that each of the two I-loaders is controlled by an individual agent. As in the reference experiment, both loaders are capable of driving to all workstations and interacting with them. The epsilon decay rate, λ_ϵ, is set to 0.993 11 during the training, resulting in the minimum epsilon, ϵ_{min}, being reached within 1000

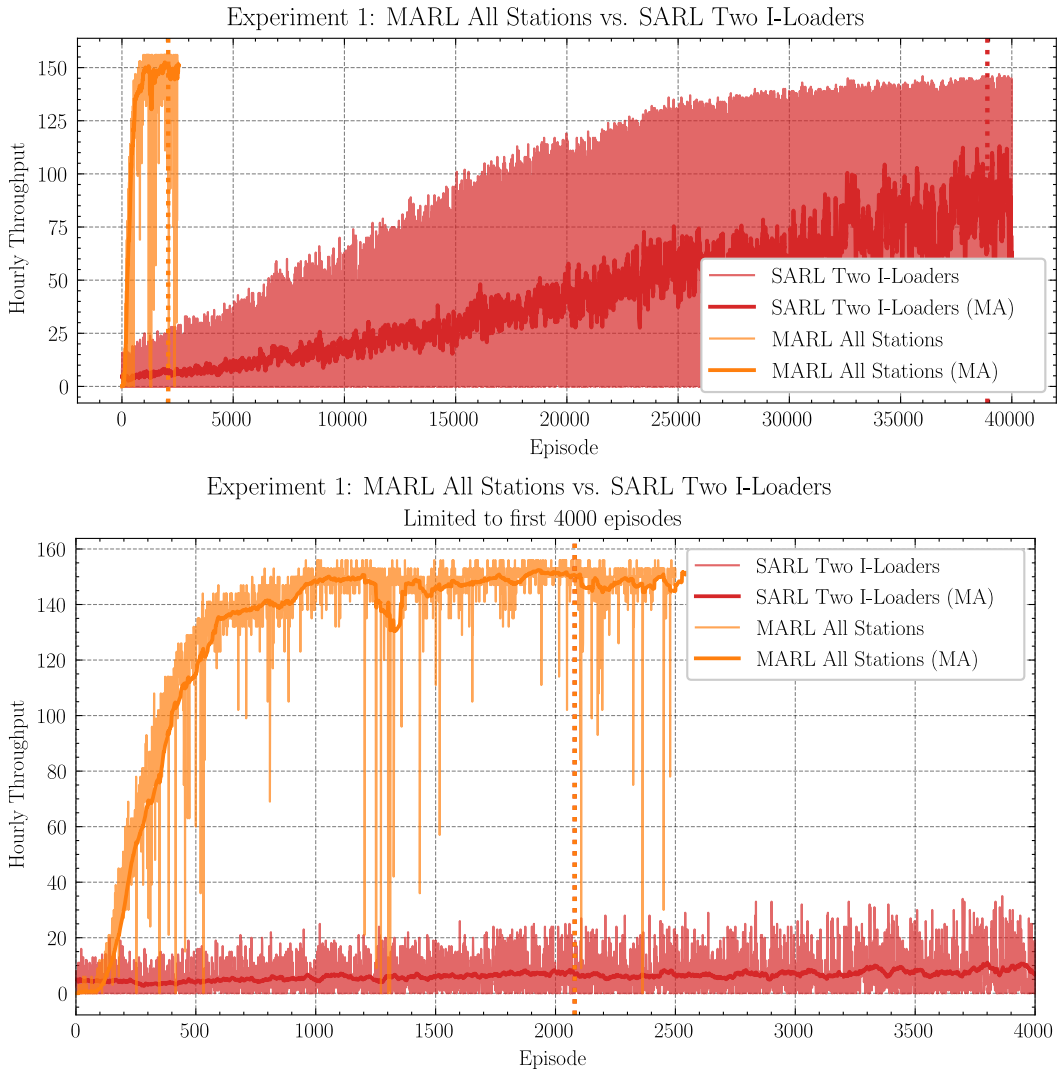

Figure 7.5: Hourly throughput of new MARL approach training run with SARL training run as comparison. To enhance visibility, a moving average is calculated over 50 episodes. The vertical dotted line denotes the episode of persistence of the "best" model. The second figure presents the plot of the same data but with the number of episodes limited to 4000.

episodes. As a consequence of the enhancements introduced by the MARL approach, the number of training episodes can also be reduced to 2500, a value comparable to the training of a model with just a single loader.

Figure 7.5 illustrates the training progress of the presented model in comparison to the reference. For improved clarity, an additional magnified version is provided, displaying only the initial 4000 episodes. The enhancement of the MARL approach is irrefutably evident. On the one hand, the maximum throughput is only reached in a fraction of the episodes. On the other hand, even a higher maximum throughput is also achieved with 156 parts per hour instead of 147 with the SARL approach. A closer examination of the zoomed plot reveals that the throughput achieved with MARL training demonstrates a markedly reduced variance in comparison to the SARL approach. The occurrence of isolated peaks or the temporary decrease around training episode

Table 7.3: Results of the validation of the "best" model for the reference strategies. All values are presented on an hourly basis.

Strategy	Throughput	Actions	
		Agent 1	Agent 2
FIFO	131		
SARL Two I-Loaders	153	2600	
MARL All Stations	158	1530	1350

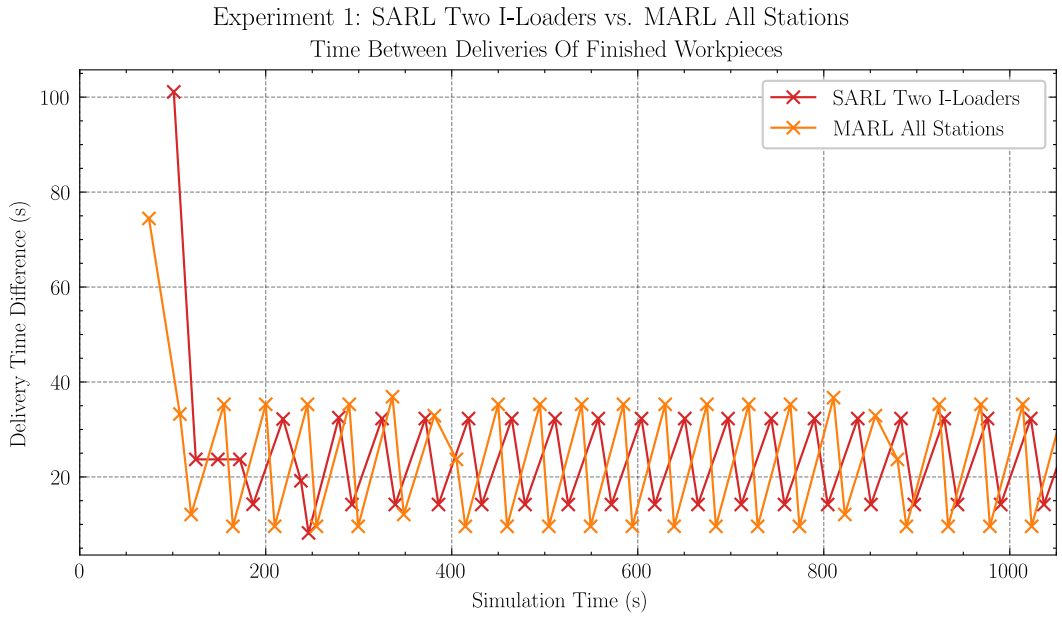

Figure 7.6: Comparison of the time between the delivery of finished workpieces at the *Conveyor_Out* station.

1300 may be caused by a number of different factors. For instance, the occurrence of isolated peaks may be attributed to the selection of an inadequate action by the ϵ-greedy approach, which results in a state that contravenes the learned cooperative strategy. An additional potential explanation is that the experiences selected for the training data (see Section 6.2.2) result in a temporary decrease in the precision of the Q-value approximations made by the DQN. This may result in a situation in which, for example, the *wait* action is consistently selected in a state, thereby causing the agent to remain in that local state. If this or a similar situation arises with one of the two agents, it halts the joint strategy and hinders or decelerates the production of further workpieces. As the training of the DQNs also occurs during the episode, these issues are mostly resolved in a few episodes.

As illustrated in Table 7.3, the results of the validation also demonstrate the enhancement of the MARL approach, as evidenced by the increased throughput. Additionally, this increased throughput is reached with approximately the same number of actions, when the numbers of both agents are combined.

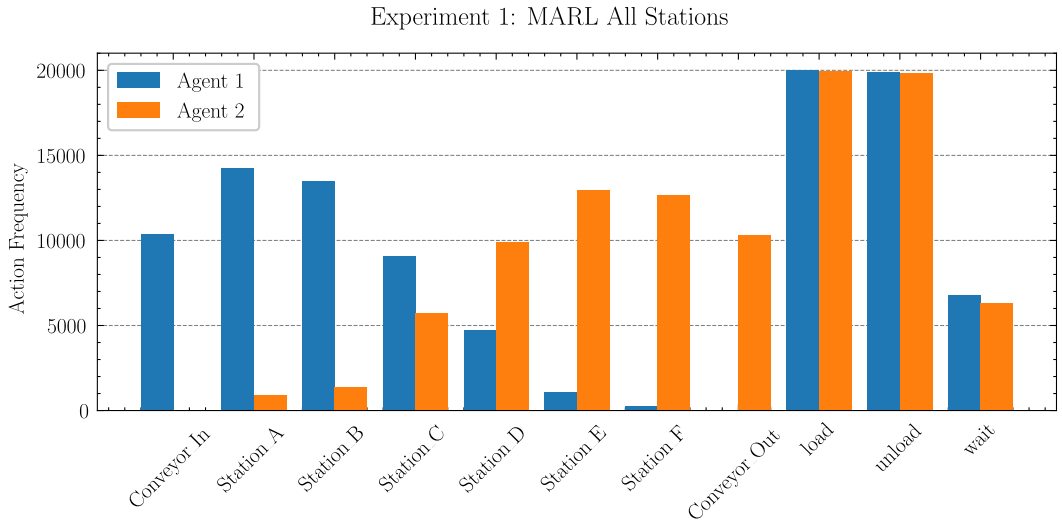

Figure 7.7: Action frequency in the replay memory of both agents of the MARL approach where all work stations can be reached by both loaders.

A closer examination of the time interval between the delivery of finished workpieces, as illustrated in Figure 7.6, reveals several noteworthy observations. Firstly, the delivery of the initial workpiece is completed at an earlier point in time, and the settling phase is shorter and follows a sequence that is more similar to the final pattern. Additionally, it is evident that the deliveries of workpieces occur at a consistent pace only for a limited duration. The observed pattern undergoes a brief alteration approximately every 500 seconds, which suggests that the learned strategy has not yet reached an optimal state. An analysis of the validation run has revealed that at these times, the loaders undertake a workpiece transport between stations that could be performed with greater efficiency by the other loader. To illustrate, if the first loader transports a processed workpiece from *Station_D* to *Station_F*, the second loader must perform an evasion drive to the *Conveyor_Out* station. Subsequently, the first loader is required to drive the extended distance back to the front area. A similar situation is observed when the second loader loads a processed workpiece from either *Station_A* or *Station_B*.

Figure 7.7 illustrates the frequencies of the various actions in the replay memory of the two agents. It is evident that these extended journeys occur with less frequency than the other trips. Furthermore, the frequencies of the driving actions of both agents exhibit a certain degree of mirroring, indicating that the learned joint strategy has independently divided the work area into two segments. This, along with the long trips to the more distant stations, which reduce throughput, reinforces the idea of restricting the workspace in advance (see Section 5.3.1). The impact of this restriction will be examined in the next experiment (see Section 7.4).

Upon examination of the relative frequency of the actions in the replay memory, it becomes evident that the *load* action and *unload* action each account for approximately 20 %. The *wait* action accounts for approximately 6 % or 7 %, depending on the agent, leaving just over 50 % for the driving actions collectively. In an optimal scenario, apart from the *wait* action, each drive to a station should be followed by a *load* or *unload* action. In other words, in an optimal strategy, the number of driving actions in an agent's

replay memory should be approximately equal to the sum of the *load* and *unload* actions. As the driving actions are represented with greater frequency in this experiment, this serves to confirm the issue of consecutive drive actions, which are challenging to avoid with the default settings. The impact of the solution presented in Section 5.3.3, which penalizes consecutive drives, is investigated in the fourth experiment (see Section 7.6).

7.4 EXPERIMENT 2: RESTRICTED WORK AREA

As observed in the preceding experiment, both agents learned a joined strategy wherein the production line is divided into two nearly distinct work areas. This experiment now examines the behavior of the training progress and the effectiveness of the learned strategy when the division is determined in advance. For that purpose, the work area of the first loader is restricted to the station from *Conveyor_In* to *Station_D*, and for the second loader from *Station_C* to *Conveyor_Out*. This restriction has the effect of reducing the local action space of both agents, as the drive actions for non-reachable stations are no longer necessary. Conversely, the local state space of each agent is also reduced, as only the features of the reachable work stations need to be present in the local state (see Section 5.3.1).

An additional restriction is introduced that penalizes the *loading* action when the next work step of the workpiece is performed at a station outside the designated work area of the loader. With reference to the production line, this restriction is intended to prevent the first loader from picking up processed workpieces from *Station_C* and *Station_D*. In the absence of this restriction, the loader would be unable to unload the workpiece, thereby preventing the acquisition of further meaningful experiences within the current training episode. This restriction has no effect on the second loader.

Figure 7.8 illustrates the training progress in comparison to the previous experiment using the hourly throughput as an indicator. It can be observed that the learning effect initiates approximately 50 episodes earlier. This outcome is attributed to the smaller local action spaces of the agents, as the restriction increases the probability that the ϵ-greedy approach is selecting a productive action actions. Upon closer examination, it becomes evident that the maximum hourly throughput is now slightly lower than that achieved with the previous setting. This reduction can be explained by the following reasons: Given that the training episodes are only 20 minutes in duration, the actual throughput achieved during the episode is only a third, in this case, 51 finished workpieces for the "Restricted Work Area" strategy and 52 workpieces for the "All Stations" strategy. As the second loader is capable of operating at *Station_A* and *Station_B* in the previous experiment, it can be productive at an earlier point in the settling phase at the beginning of the episode. This slight acceleration of the settling phase allows both agents to proceed with the actual strategy more quickly, enabling the delivery of another finished workpiece within the 20 minutes of the episode. Nevertheless, this acceleration is effectively negated during the validation process, as the settling phase is relatively brief in comparison to the overall duration of the validation.

Figure 7.9 illustrates the hourly throughput of the validation for both the "best" and the "final" model of the "Restricted Work Area" strategy. It is noteworthy that the

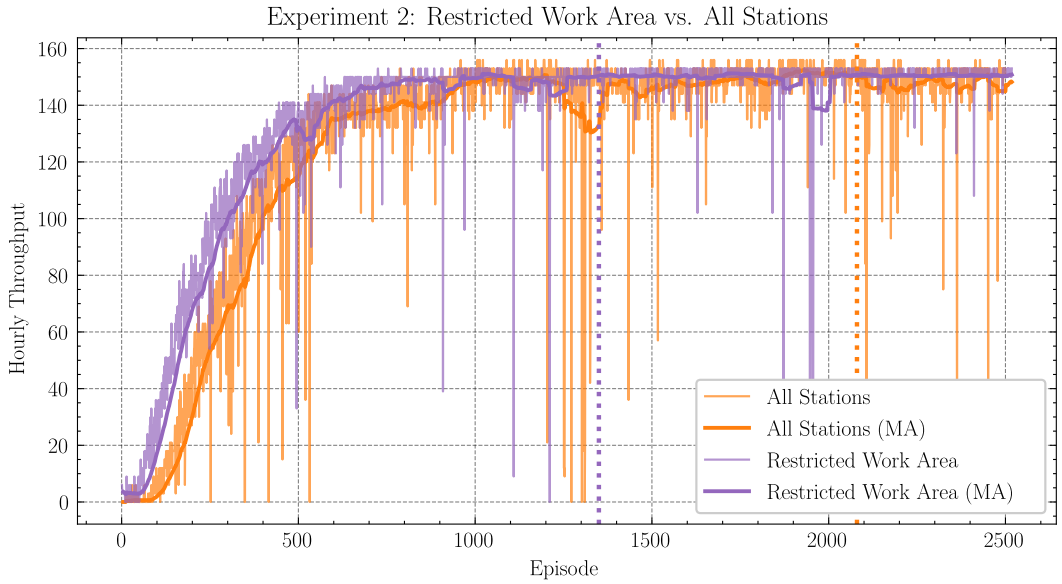

Figure 7.8: Hourly throughput of the training run with restricted work area in comparison where all stations can be reached. To enhance visibility, a moving average is calculated over 50 episodes. The vertical dotted line denotes the episode of persistence of the "best" model.

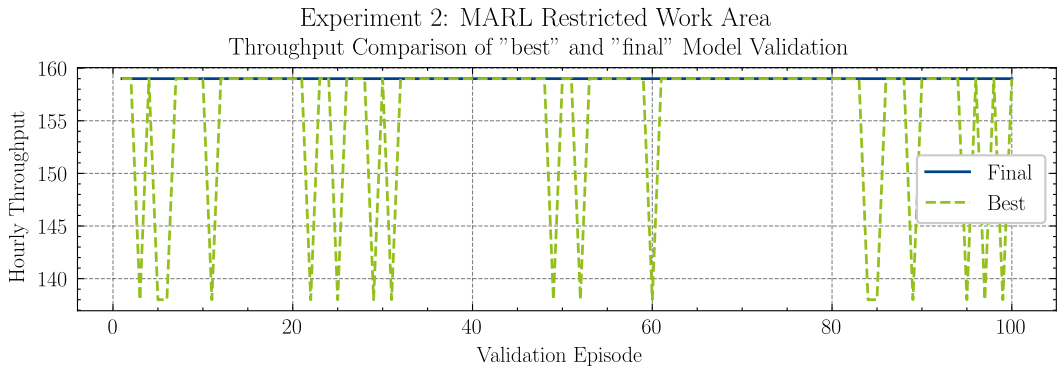

Figure 7.9: Comparison of the reached hourly throughput of the "best" and "final" model for the "Restricted Work Area" strategy.

Table 7.4: Results of the validation of the "best" model for the reference strategies. All values are presented on an hourly basis.

		Actions	
Strategy	**Throughput**	**Agent 1**	**Agent 2**
FIFO All Stations	131		
MARL All Stations	158	1530	1350
MARL Restricted Work Area			
best (max)	159	1520	1360
best (min)	138	1390	1400
final	159	1750	1450

"best" model generates a markedly reduced throughput in several episodes delivering only 138 finished workpieces. This inconsistency can be derived from the relatively early persistence of the "best" model in the 1350th training episode (see Figure 7.8). It can be hypothesized that a higher throughput was achieved by chance over several episodes, which then resulted in the persistence of the model. However, due to the early persistence, further fine-tuning of the DQN that probably happened in the following episodes did not persist, as the maximum throughput was already reached. Accordingly, the "final" model, which resembles the model after the completion of all 2500 training episodes, is validated to ascertain whether the fine-tuning process was indeed successful. As illustrated in Figure 7.9, the "final" model consistently achieves a throughput of 159 workpieces per hour, which represents an improvement compared to the first experiment. However, when examining the actions required for this throughput (see Table 7.4), it is evident that the "final" model needs approximately 300 additional actions compared to the best model in a "good" episode. The findings indicate that extended training periods may not always yield positive outcomes.

7.5 EXPERIMENT 3: GRADUATED REWARD

As part of this third experiment, an adjustment has been made to the reward function. This adjustment is made with the aim of encouraging the rapid transfer of workpieces through the production line (see Section 5.3.3). Previously, the *unloading* of a workpiece into a work station or the *Conveyor_Out* was rewarded with a fixed value. However, the reward is now increasing, depending on the process step of the workpiece. This results in the following reward function (Equation 7.2) for the production line in question:

$$R_{t+1} = \begin{cases} 4, & \text{unloading a finished workpiece at } \textit{Conveyor\_Out} \\ 3, & \text{unloading a workpiece at } \textit{Station\_E} \text{ or } \textit{Station\_F} \\ 2, & \text{unloading a workpiece at } \textit{Station\_C} \text{ or } \textit{Station\_D} \\ 1, & \text{unloading a workpiece at } \textit{Station\_A} \text{ or } \textit{Station\_B} \\ 1, & \text{loading a workpiece from } \textit{Conveyor\_In} \text{ or a work station} \\ -3, & \text{waiting at a station while blocking other loader} \\ 0, & \text{otherwise} \end{cases} \tag{7.2}$$

Figure 7.10 illustrates the training progress in comparison to the training of the previous experiment. It is noteworthy that the reward function adjustment does not result in a significant enhancement of the training process. However, a slight reduction in variance in the achieved throughput and a reduction in the number of temporary drops can be observed. Additionally, it is evident that the "best" model has persisted at a later episode in comparison to the previous experiment. However, these observations may also have resulted from chance and thus should be interpreted with caution.

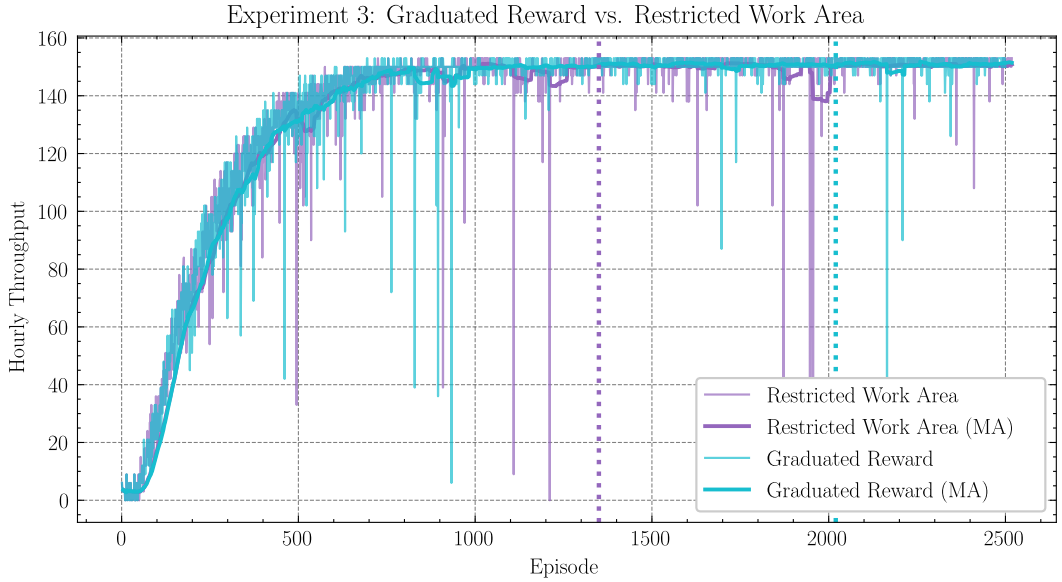

Figure 7.10: Hourly throughput of the training run which uses the graduated reward in comparison where the work area is restricted. To enhance visibility, a moving average is calculated over 50 episodes. The vertical dotted line denotes the episode of persistence of the "best" model.

Table 7.5: Results of the validation of the "best" model of the "Graduated Reward" strategy with the results of the "Restricted Work Area" strategy as reference. All values are presented on an hourly basis.

		Actions	
Strategy	**Throughput**	**Agent 1**	**Agent 2**
MARL Restricted Work Area			
best (good)	159	1520	1360
best (bad)	138	1390	1400
final	159	1750	1450
MARL Graduated Reward	159	1600	1450

As demonstrated in Table 7.5, the validation results of the "best" model indicate that the maximum hourly throughput was once again achieved. In comparison to the "final" model of the "Restricted Work Area" strategy the number of actions performed by the second agent remained constant. In contrast, the first agent was able to identify an improved local strategy in terms of the required actions. However, in comparison with the "best" model in a "good" validation episode, this does not yet appear to be optimal. Nevertheless, caution should be exercised when making such comparisons, as the throughput that triggered the persistence of the "best" model is achieved by the joint strategy of both agents. Consequently, the local strategies of the agents could potentially differ even with the same number of actions and reached throughput.

Figure 7.11 illustrates the frequencies of the actions in the replay memory of both agents from the training of the "Graduated Reward" strategy. It is evident that the second agent has a notably lower frequency of *wait* actions within the replay memory, thereby indicating that all other actions are represented with slightly greater frequency than with the first agent. This discrepancy may be attributed to the slightly longer travel time of two seconds between *Station_F* and the *Conveyor_Out* station, as opposed to the opposite drive of the first agent between *Station_A* and the *Conveyor_In* station, which takes only 1.5 seconds (see Table 7.1). The longer travel time for the outward and return drive of two times half a second appears to result in a reduction of necessary *waiting* actions.

It is also noteworthy that the number of drive actions to the two work centers, which can only be reached by a single agent, is approximately equivalent. With regard to the inner work center (comprising of *Station_C* and *Station_D*), which has been identified as the critical zone for blockages, it is determined that the rear station is only approached approximately half as often. This implies that the first agent approaches *Station_C* twice as frequently as *Station_D* and vice versa for the second agent. One potential explanation for this situation is the possibility of evasion drives. To illustrate, if the second loader is located at *Station_C* and the first loader desires to transport a workpiece to *Station_C*, the second agent will probably perform a brief evasion journey to *Station_D*. In this situation, it would be more efficient to drive directly to another work station where a workpiece can be loaded or unloaded, rather than making an evasion drive to the next station that is mostly followed by a consecutive drive. The following experiment examines whether penalizing consecutive drives has an influence on this situation.

7.6 EXPERIMENT 4: CONSECUTIVE DRIVE PENALTY

The objective of this experiment is to investigate the impact of introducing a penalty for consecutive drives. The application of this penalty is justified by the fact that the simulation software in use makes it extremely challenging for the agents to differentiate between consecutive drives and more efficient direct drives. This phenomenon is a consequence of the fact that the travel times between stations adhere to a constant value and fail to reflect the acceleration and deceleration observed in reality (see Section 5.3.3).

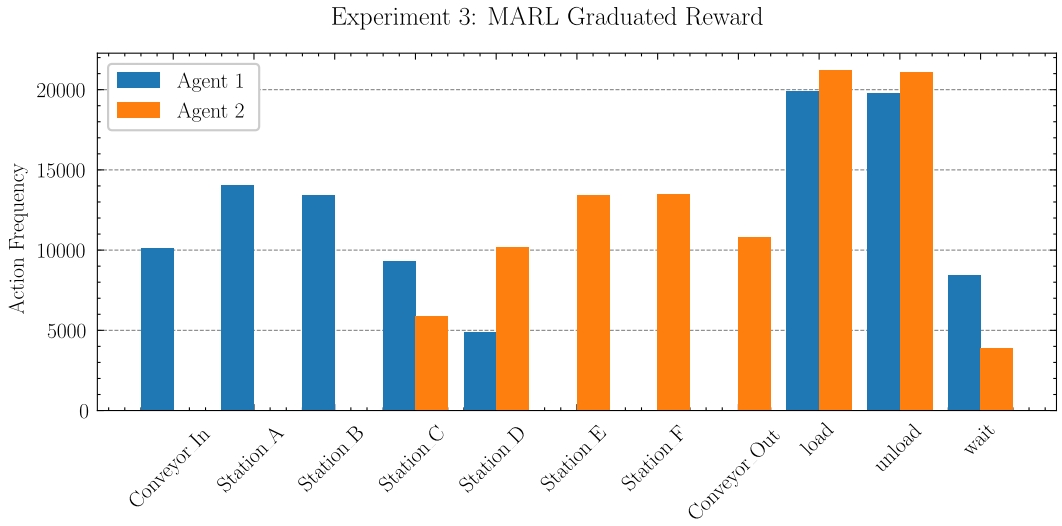

Figure 7.11: Frequency of the distinct action in the replay memory of both agents.

The reward function from the previous experiment (see Equation 7.2) is extended by the following aspect:

$$R_{t+1} = \begin{cases} \dots \\ -1, & \text{if the actions } a_t \text{ and } a_{t-1} \text{ are both drive actions} \\ \dots \end{cases} \tag{7.3}$$

Figure 7.12 illustrates the training progress in comparison to that of the previous experiment. It is evident that the training progress has become more diverse, exhibiting a greater degree of variability and notable occurrences of temporary declines. This lends support to the hypothesis put forth in Section 5.3.3 that the introduction of the penalty results in different reward values being attributed to the same state-action-state transition. Notwithstanding this disadvantage, the same maximum throughput is achieved during the training.

As illustrated in Table 7.6, the results of the validation are presented in comparison to the outcomes of the previous experiment. It is noteworthy that the "best" model once again achieved the maximum throughput of 159 workpieces per hour. The introduction of the penalty has resulted in a reduction of the number of necessary actions from approximately 3000 to 2800.

However, the validation of the "final" model nearly produced no workpiece. The "final" model represents the state of both agents after the completion of all 2500 training episodes. As seen in Figure 7.12 the training completed within a drop in the achieved throughput. Further analysis of the validation episode has shown that the second agent wrongfully prefers the *wait* action in a state where processed workpieces could have been transported to the next station. This may only be a temporary problem caused by the randomly selected experiences that are used for the training of the DQN.

An examination of the frequency of the actions in the replay memory of both agents also demonstrates that the consecutive drives are successfully avoided (see Figure 7.13).

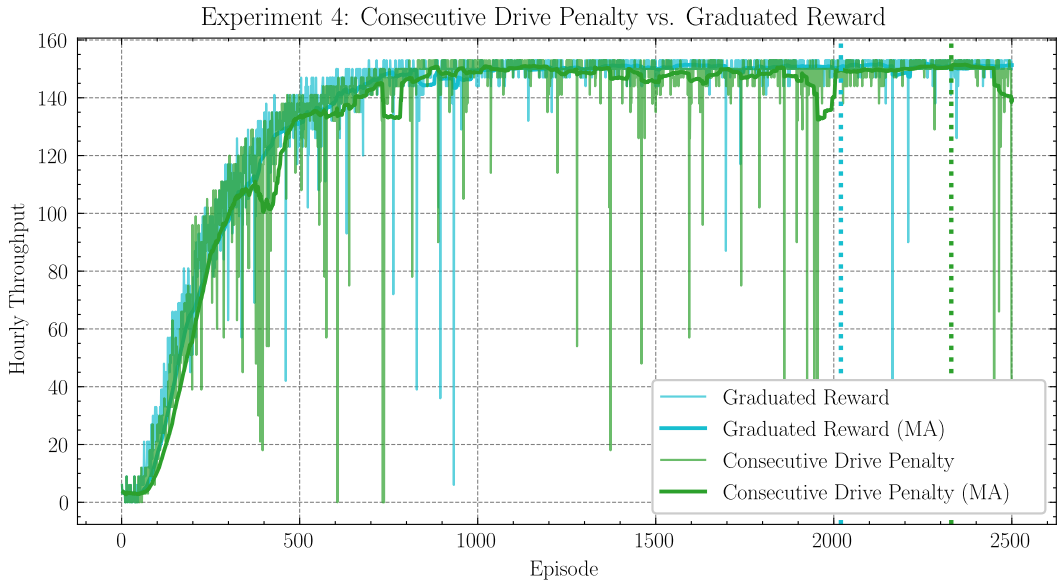

Figure 7.12: Hourly throughput of the training run which employs the penalty for consecutive drives in comparison with the training run of the third experiment. To enhance visibility, a moving average is calculated over 50 episodes. The vertical dotted line denotes the episode of persistence of the "best" model.

Table 7.6: Results of the validation of the "best" model of the "Consecutive Drive Penalty" strategy with the results of the "Graduated Reward" strategy as reference. All values are presented on an hourly basis.

		Actions	
Strategy	**Throughput**	**Agent 1**	**Agent 2**
MARL Graduated Reward	159	1600	1450
MARL Consecutive Drive Penalty			
best	159	1510	1290
final	2	120	200

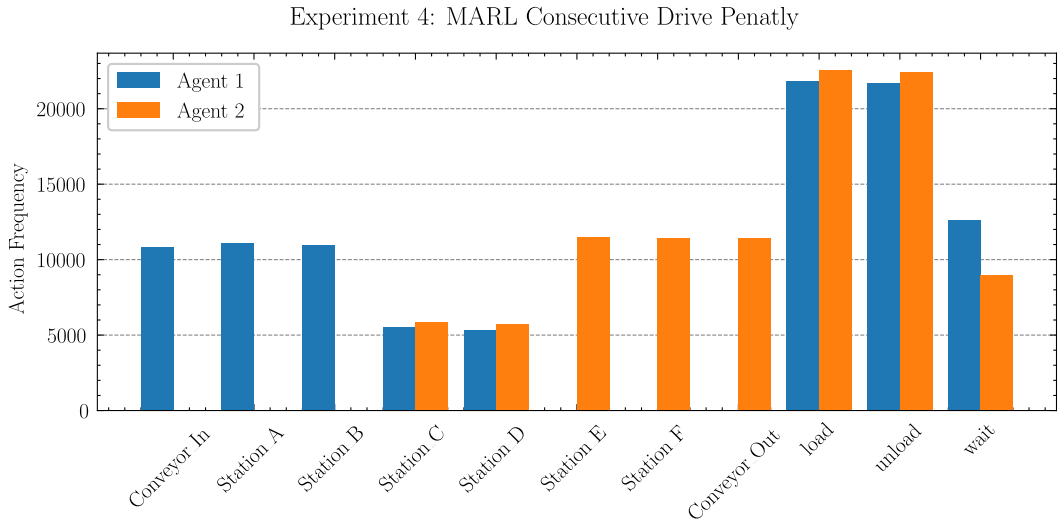

Figure 7.13: Frequency of the distinct actions in the replay memory of both agents.

It can be observed that all stations are now approached with approximately equal frequency. *Station_C* and *Station_D* represent an exception with regard to a single agent, as on the one hand, the first agent is only approaching for the purpose of loading the stations, and on the other hand, the second agent is only approaching for the purpose of unloading. The other work stations are approached by the same agent for both processes. When the frequencies of both agents are summed, the result is a value that is similar to the frequency of the other drive actions.

Furthermore, it can be observed that the sum of all driving actions of an agent now occurs with a frequency that is nearly equivalent to the sum of the *loading* and *unloading* actions. This lends further support to the hypothesis that the ideal strategy is for each drive to be followed by a *loading* or *unloading* action. It is also possible that *wait* actions occur between two drive actions, as this is not detected by the penalty mechanism and therefore not penalized. Nevertheless, it can be postulated that this occurs with minimal frequency, as otherwise the ratio between the number of driving and loading/unloading actions would be skewed towards the driving actions.

7.7 EXPERIMENT 5: MACHINE FAILURES

In the preceding experiments, all processing times on the machines were held constant, thereby establishing the theoretical premise that the optimum strategy represents the most efficient driving pattern. In practice, however, malfunctions may occur on the machines, which then disrupt that pattern. It is imperative that the agent's joint strategy also yields the best possible throughput in such scenarios. In order to evaluate the behavior of the different strategies, machine failures are simulated in this experiment. A machine failure can only occur after a workpiece has been loaded into a work machine. The error itself is flagged within the state space with the "failed" feature of the corresponding machine (see Table 5.6). As long as the machine is faulty, it is not possible for any loader to interact with it. Once the error has been resolved, the machine

switches to the idle state, and the previously processed workpiece is destroyed, thereby disappearing from the simulation.

In practice, the Mean Time Between Failures (MTBF) and Mean Time to Repair (MTTR) describe the availability (Equation 7.4) of machines, which is defined as follows:

$$Availability = \frac{MTBF}{MTBF + MTTR} \tag{7.4}$$

In this experiment, the MTBF is set to 270 seconds, and the MTTR is set to 30 seconds, resulting in a machine availability of 90 %. This relatively low availability is chosen in order to illustrate the impact of machine failures on training more clearly. In practice, errors do not occur at fixed intervals; rather, they occur randomly. Accordingly, the MTBF and MTTR values serve as parameters for statistical distributions, which are employed to determine the simulated values in the simulation. The time to the next failure is determined using the exponential distribution with the parameter $\lambda = {}^{1}/_{MTBF}$. The exponential distribution is defined by the following density (Equation 7.5) and the distribution function (Equation 7.6):

$$f(x) = \begin{cases} \lambda \exp(-\lambda x) & \text{for } x \geq 0 \\ 0 & otherwise \end{cases} \tag{7.5}$$

$$F(x) = 1 - \exp(-\lambda x) \tag{7.6}$$

In the case of MTBF, only the active processing time of the machines is considered. To illustrate, if an MTBF of 255 seconds is determined for a machine, this signifies that the error occurs when the 26$^{\text{th}}$ workpiece is processed, given that the processing of each workpiece takes 10 seconds.

The repair times of the machines are determined using the Erlang distribution, with the parameters $\lambda = \frac{1}{2 \times MTTR}$ and $k = 2$. The Erlang distribution is defined by the following density function (Equation 7.7) and the distribution function (Equation 7.8):

$$f(x) = \begin{cases} \frac{\lambda^k x^{k-1} \exp(-\lambda x)}{(k-1)!} & \text{for } x \geq 0 \\ 0 & otherwise \end{cases} \tag{7.7}$$

$$F(X) = 1 - \sum_{n=0}^{k-1} \frac{1}{n!} \exp(-\lambda x)(\lambda x)^n \tag{7.8}$$

Upon completion of the repair process, new values for the MTBF and the MTTR are generated using the aforementioned distributions, which subsequently describe the future occurrence of machine failures. It is important to note that the probability of simultaneous failure of both machines in a work center is 1 %, as the failures of the machines are independent and identically distributed.

In order to test the resilience of the MARL approach to machine failures, all configurations from the previous experiments were retrained with the aforementioned machine

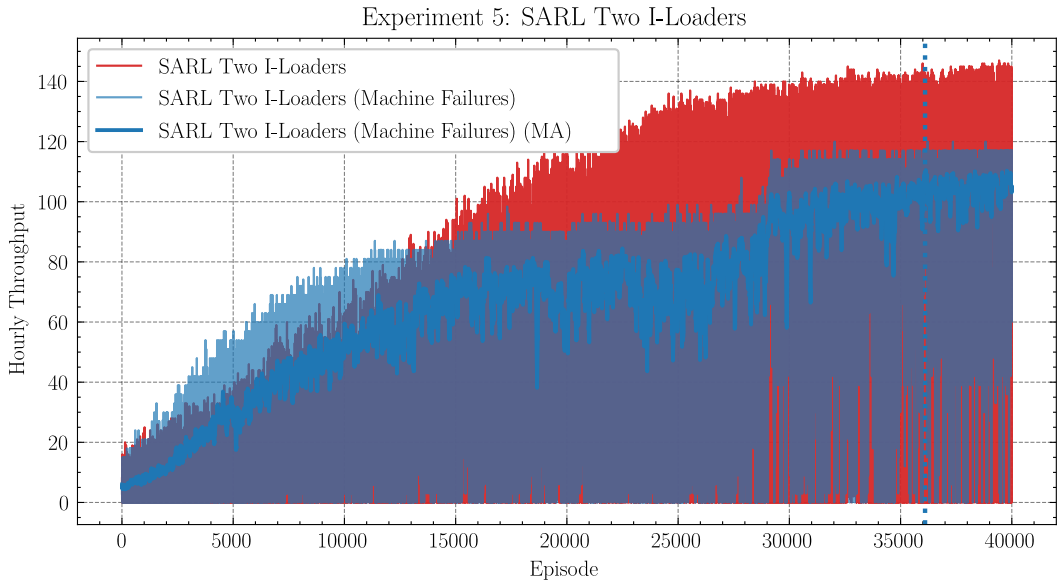

Figure 7.14: Hourly throughput achieved during training with the SARL implementation. To enhance visibility, a moving average is calculated over 50 episodes.

failures. Figure 7.14 illustrates the training progress of the SARL approach in comparison to the same training without failures. As can be observed, the training process is not particularly effective, exhibiting a stagnation at approximately 120 parts per hour. The validation results, as presented in Table 7.7, also corroborate this conclusion. Due to the occurrence of simulated machine failures, the throughput achieved and the number of actions performed exhibited considerable fluctuations over the course of the 100 validation episodes. Consequently, the table presents the 95 % confidence interval for the mean of the different metrics across the validation episodes. The "final" model that underwent a slightly longer training period demonstrated marginally better performance. The suboptimal performance can be attributed to multiple factors, one of which is the increase in state space cardinality due to the introduction of the "failed" state for each machine. Furthermore, it has been demonstrated once again that the moving average of the throughput is not an optimal metric for the persistence of the "best" model.

Figure 7.15 illustrates the progression of the MARL training runs in two pairs. A comparison between the "All Stations" and "Restricted Work Area" strategies demonstrates a more rapid increase, similar to that observed in the second experiment (see Section 7.4). However, in contrast to the previous experiment, the "Restricted Work Area" strategy appears to yield superior training outcomes when machine failure are simulated.

A comparison of the "Graduated Reward" and "Consecutive Drive Penalty" strategies reveals that the training progress is again more variable due to the introduction of consecutive drive penalty as before in the fourth experiment (see Section 7.6). All four training runs exhibit a comparable maximum, with an output of approximately 140 workpieces per hour. In comparison to all previous experiments, each resulting training progress displays greater variability, which can be attributed to both the inherent unpredictability of the failures themselves and the slightly larger state space. To ensure the consistency of the training runs, the same hyperparameters were intentionally used.

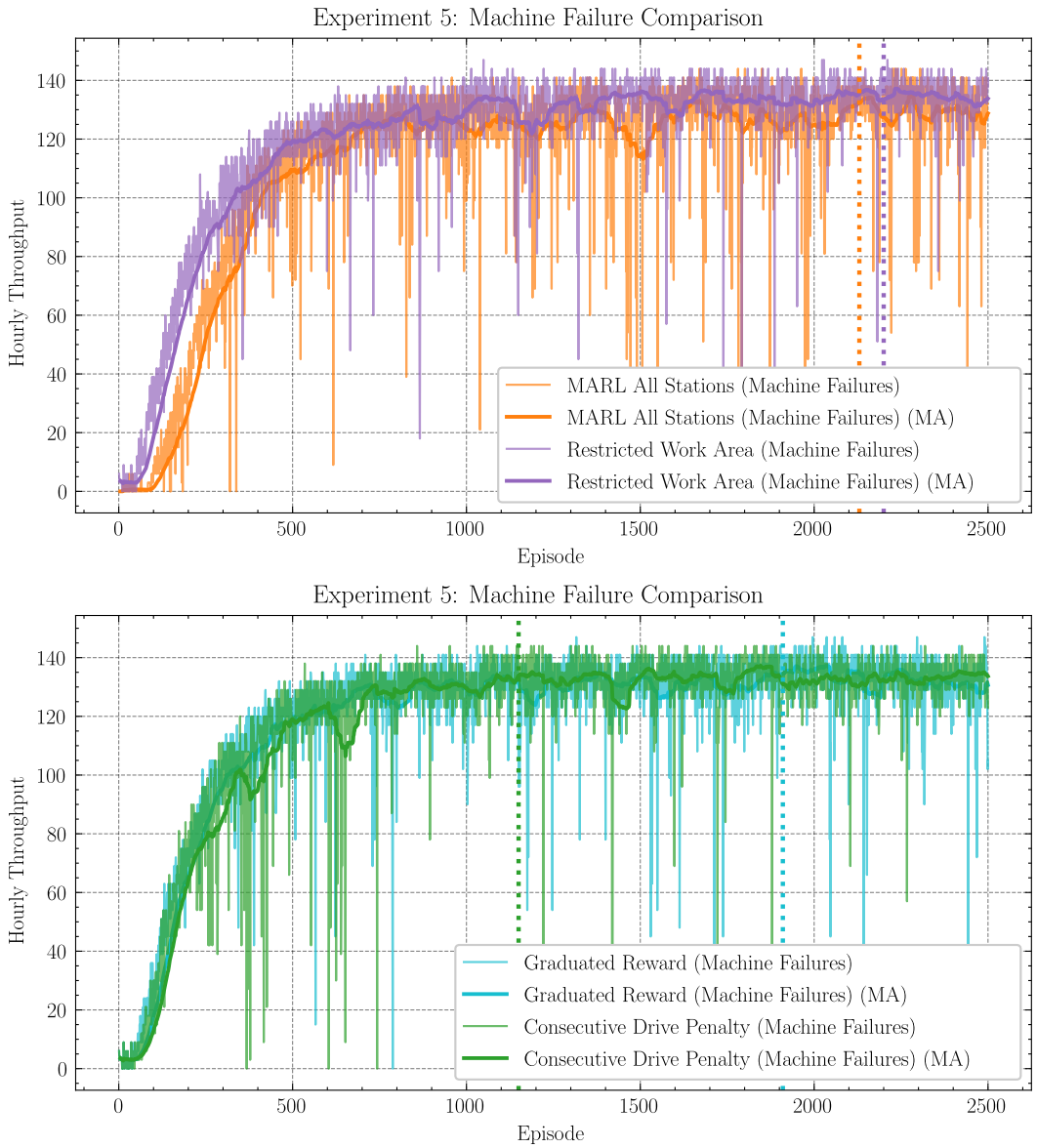

Figure 7.15: Hourly throughput during training of all four MARL strategies experiments with machine failures. To enhance visibility, a moving average is calculated over 50 episodes.

It is likely that the training runs would exhibit less variability if the epsilon decay rate was slower, more training episodes were conducted, or the simulated time was extended.

A closer examination of the validation results reveals that while the "best" models exhibit similar throughputs, the MARL strategies still demonstrate notable differences. The "final" models of the "All Stations" and "Graduated Reward" strategies exhibit a notable decline in performance relative to their "best" models. This discrepancy can be attributed to the inherent randomness associated with the training process. The "Restricted Work Area" strategy achieves the highest throughput with both models, with minimal variance. An examination of the frequencies of the various actions in the replay memory (see Figure 7.16) reveals that even in the presence of machine failures, the punishment of double runs has an effect.

Additionally, it is evident that the second agent awaits almost twice as frequently as the first loader. This is attributed to the fact that the initial failures predominantly occur at the front stations, as these also commence processing first. Consequently, due to the destruction of the workpiece by the machine failure and the subsequent repair time, which averages 30 seconds, only one machine is operational for the stations situated behind it during this interval. This results in a notable increase in idle time for the second loader.

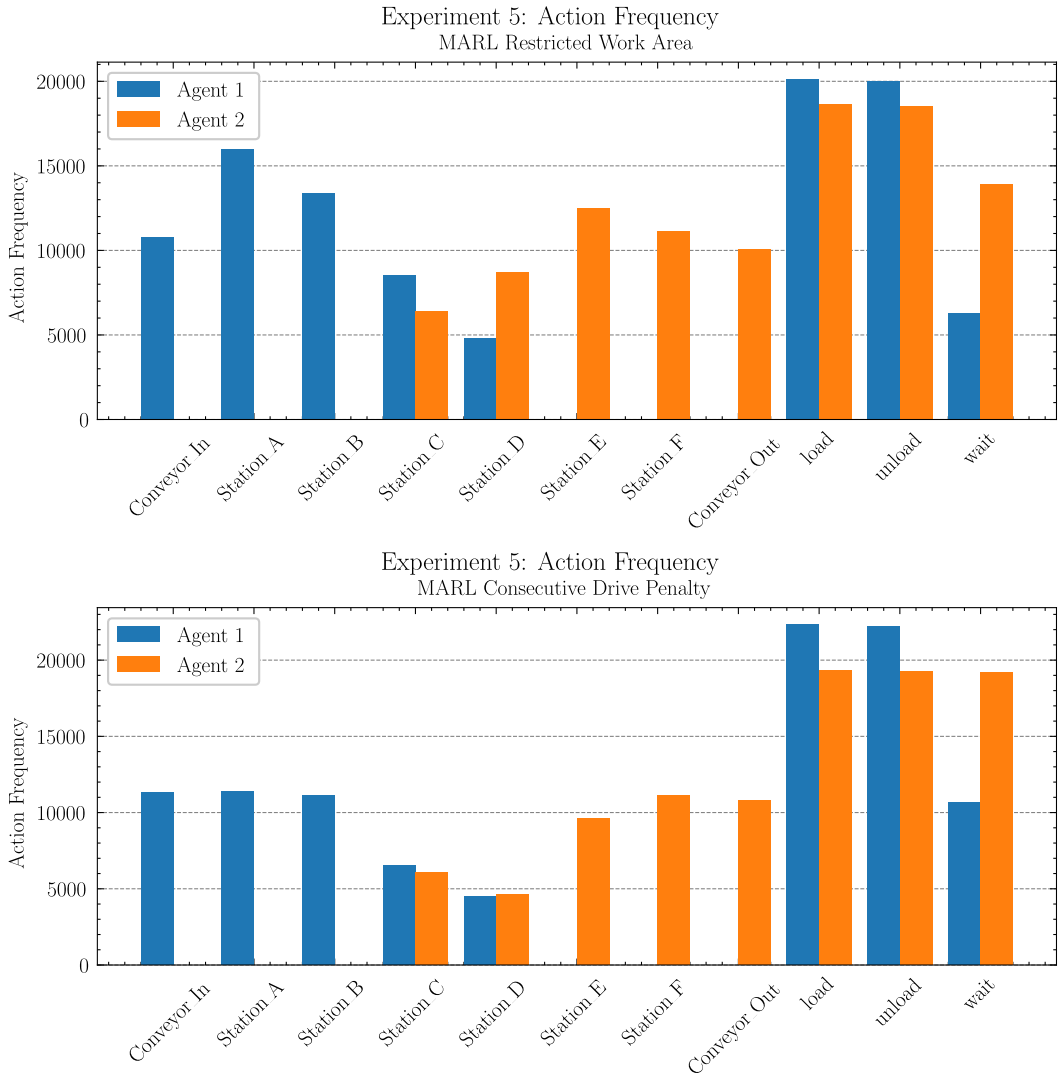

Figure 7.16: Frequency of the action in the replay memory from the training runs of the "Restricted Work Area" and the "Consecutive Drive Penalty" Strategy.

Table 7.7: Results of the validation runs for the different strategies and models when machine failures ($MTBF = 270\,\text{s}, MTTR = 30\,\text{s}$) are simulated. The results are presented as the mean over all 100 validation episodes with the 95 % confidence interval. All values are presented on an hourly basis.

		Actions	
Strategy	**Throughput**	**Agent 1**	**Agent 2**
FIFO All Stations	124.5 [124.07, 124.86]		
FIFO Restricted Area	114.3 [113.71, 114.96]		
SARL Two I-Loaders			
best	116.2 [112.2, 115.54]	2162.7 [2128.75, 2196.56]	
final	112.2 [114.15, 118.28]	2110.3 [2056.77, 2163.81]	
MARL All Stations			
best	135.5 [131.47, 139.51]	1616.4 [1603.69, 1629.21]	1434.0 [1422.53, 1445.57]
final	110.0 [100.45, 119.59]	1386.3 [1297.01, 1475.54]	1421.8 [1329.28, 1514.37]
MARL Restricted Work Area			
best	142.8 [142.29, 143.23]	1526.5 [1524.43, 1528.64]	1423.0 [1419.46, 1426.63]
final	144.5 [143.98, 145.01]	1658.0 [1655.94, 1659.97]	1427.4 [1424.85, 1429.88]
MARL Graduated Reward			
best	141.0 [140.45, 141.51]	1565.3 [1561.34, 1569.19]	1448.0 [1444.12, 1451.88]
final	125.3 [118.8, 131.73]	1819.8 [1765.92, 1873.74]	1726.6 [1667.91, 1785.23]
MARL Consecutive Drive Penalty			
best	135.7 [131.37, 140.12]	1425.8 [1415.75, 1435.9]	1341.9 [1334.38, 1349.36]
final	134.9 [134.43, 135.42]	1353.6 [1351.39, 1355.89]	1379.5 [1376.78, 1382.25]

Table 7.8: Adjusted configuration for both training strategies.

Strategy	# Episode	Simulated Time	Epsilon Decay Rate
Five Steps	2500	1200 s	0.993 11
Five Steps Slowed Training	5000	1800 s	0.9977

7.8 EXPERIMENT 6: FIVE WORK STEPS

In this final experiment, the performance of the new MARL approach in a production line examined in which the workpieces must pass through five work steps. As in the preceding experiments, there are two work stations for each work step. The processing times and distances between the machines align with the established pattern. The work areas for the agents are separated at *Station_E* and *Station_F*, allowing each agent to reach six work stations. The state space of the preceding SARL implementation is estimated to have a state space cardinality of approximately 5.2×10^{13}. In contrast, the cardinality of the local state spaces in the MARL approach is observed to be 4.4×10^{7} for the first agent and 1.8×10^{7} for the second agent. These values confirm the rapid growth of the SARL state space with minor alterations, due to the necessity of mapping the complete system state.

In the interest of clarity of presentation, machine failures have been excluded from this experiment. Two distinct configurations of the training parameters are subjected to evaluation (see Table 7.8). The initial configuration employs the identical settings utilized in preceding MARL experiments. The second configuration results in a slower training process, with a reduced epsilon decay rate, thereby allowing for a more comprehensive exploration of the expanded local state space of both agents. The epsilon decay rate of $\lambda_\epsilon = 0.9977$ results in the minimum being reached after 3000 training episodes. As a result, the number of episodes is increased to 5000. Given that the higher number of process steps increases the throughput time of each workpiece, the simulated time per episode is increased to 30 minutes. This ensures that, especially at the beginning of the training, i.e., while a high epsilon value is still present, there are sufficient opportunities for workpieces to be guided through the system by random decisions only.

The progression of both training runs is illustrated in Figure 7.17. It can be observed that a higher throughput is achieved in a shorter time with the standard settings, as the proportion of decisions based on the Q-values increases at a faster rate due to the accelerated epsilon decay. However, it is apparent that there is a notable degree of variability in the achieved throughput. This is due to the fact that the strategy that each agent must learn is more complex, as each agent is now responsible for three work steps. Furthermore, the first agent is required to first guide the workpiece through the initial work steps before the second agent can gain any productive experience.

The progression of the slowed-down training exhibits a slower increase, which is attributed to the reduced epsilon decay rate. However, the variance observed during training appears to be similar to that of the default training run. A more notable observation is that a higher maximum throughput of 100 parts per hour is achieved

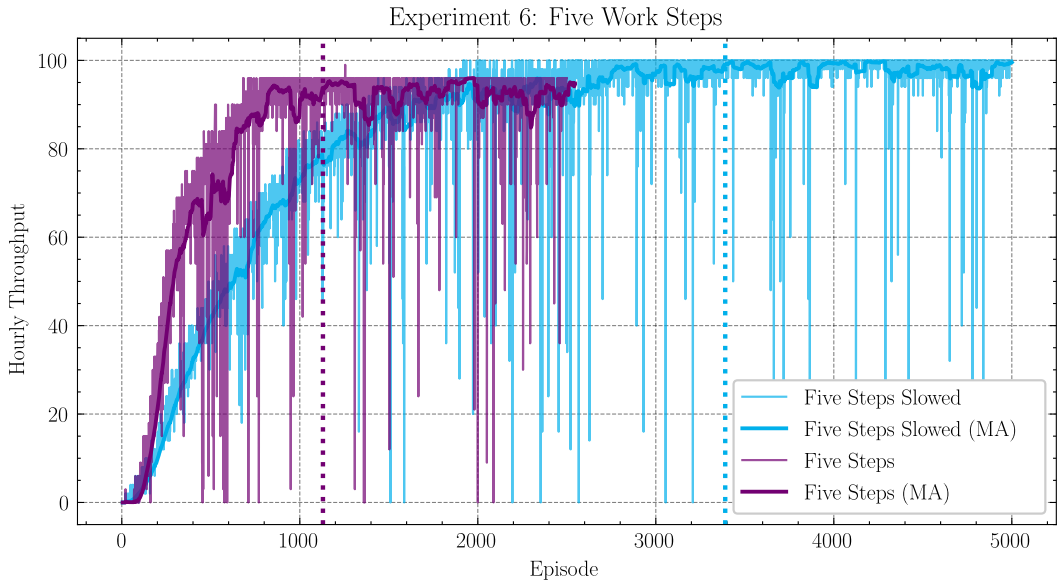

Figure 7.17: Hourly achieved throughput of the training runs of the sixth experiment. To enhance visibility, a moving average is calculated over 50 episodes. The vertical dotted line denotes the episode of persistence of the "best" model.

with the "Slowed" strategy. This discrepancy is likely attributable to the extended simulated time span, whereby the system's settling phase has a diminished impact on the attained throughput rate.

The results of the validations for both strategies are presented in Table 7.9. It can be observed that the discrepancies in both the achieved throughput and the actions required are relatively minor. The "final" model of the "Five Steps" strategy exhibits a slight decline in performance, which is attributed to random fluctuations during the training phase. The slower training process, which results in an extended exploration of the state space using the ϵ-greedy approach, resulted in a throughput of 105 per hour in approximately half of the validation episodes.

Table 7.9: Results of the validation of both models of each experiment training run. All values are presented on an hourly basis. The "Slowed Training" strategy achieved each corresponding throughput in approximately 50 validation episodes.

		Actions	
Strategy	**Throughput**	**Agent 1**	**Agent 2**
MARL Five Steps			
best	104	1370	1320
final	103	1470	1370
MARL Five Steps Slowed Training			
best	{104, 105}	1370	1320
final	{104, 105}	1400	1350

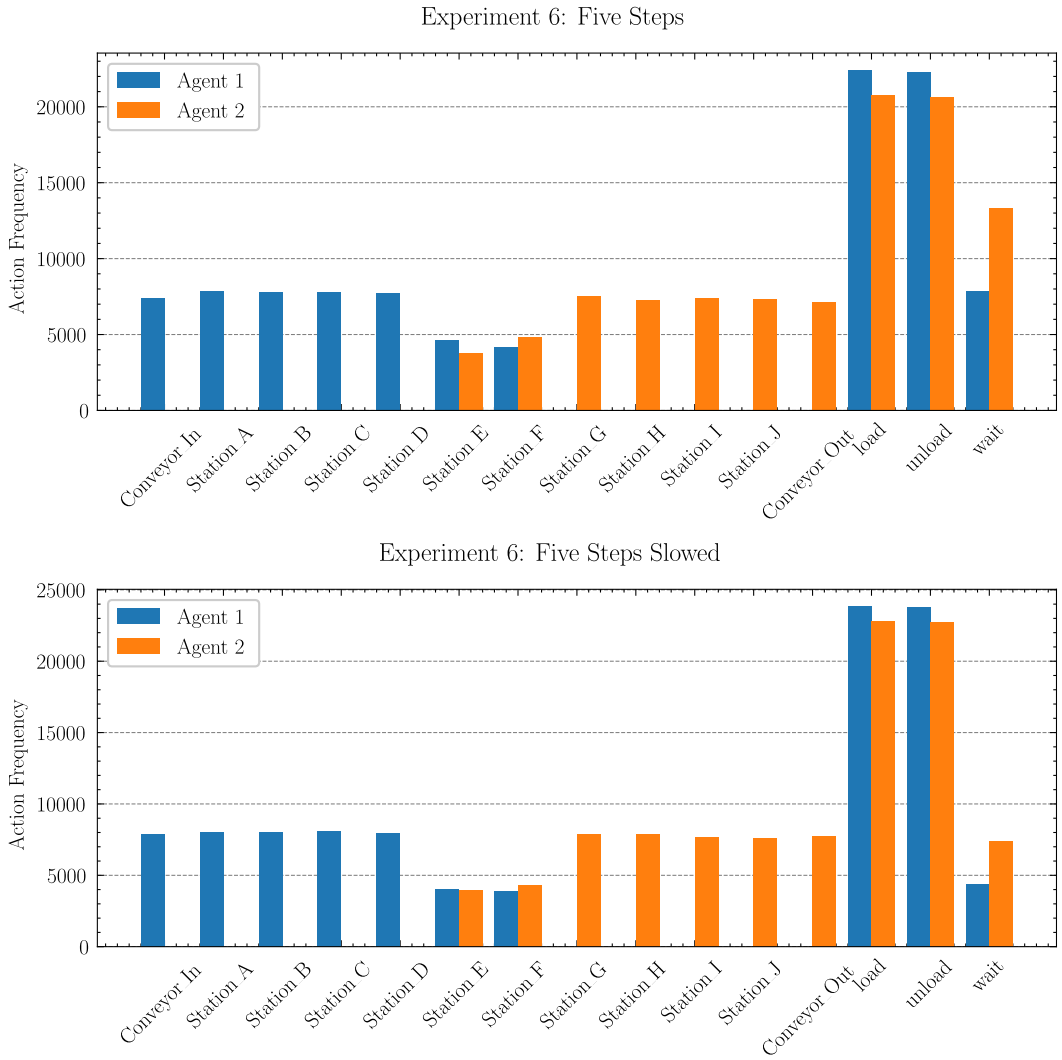

Figure 7.18: Frequency of the distinct actions in the replay memory of both agents.

Figure 7.17 once more illustrates the frequencies of the actions stored in the replay memory for both training runs. In both cases, it can be observed that the implementation of a penalty for double trips has resulted in a similar frequency of occurrence for all driving actions. Furthermore, the ratio between drive and *load/unload* actions is also balanced. As the training episodes are terminated after the specified time, the drive actions to the rear stations (*Station_G* to *Conveyor_Out*) occur with slightly diminished frequency, as the workpieces that are already in the system are no longer completed. The most notable distinction between the two training runs is the frequency of waiting actions. On the one hand, it can be observed that the training with the standard settings involves more frequent waiting. This may also be the reason for the slightly lower throughput during the validation process, as the agents may mistakenly select the *wait* action, which is aborted after a very short time due to a passive updates in the system state (see Section 5.1.1). This issue seems to have been resolved with the slower training.

Secondly, it is evident that the second agent now waits more frequently in both training runs in comparison to the previous experiments. This is due to the longer settling phase of the system. By the time the second agent can retrieve a completed workpiece from *Station_E* or *Station_F*, the first agent has already performed a substantial number of actions. As a consequence of the manner in which the simulation software operates, each of these actions constitutes a passive alteration to the system state that cancels the second agent's *wait* action (see Section 5.1.1). An enhancement to the simulation software could be the incorporation of a mechanism that considers the restricted work area of the loaders, whereby requests for the loader are only sent to the agent in the event of a change in the loader's work area.

8

CONCLUSION

In the modern era, many production facilities have adopted robots to automate tasks and improve efficiency, but traditional control methods are no longer sufficient to maintain adaptability, leading to the growing popularity of dynamic scheduling. Recent research has increasingly focused on using Artificial Intelligence (AI) to optimize dynamic scheduling, with digital twins providing a strong foundation for applying Reinforcement Learning (RL) to complex and continuous tasks.

The analysis of the preceding Single-Agent Reinforcement Learning (SARL) approach for controlling gantry robots in a production line has demonstrated that the approach is unable to adapt effectively when multiple loaders are utilized. As a consequence of the requirement of mapping the entire system state, even minor additions to the production line layout result in an exponential growth of the state space. Similarly, the asynchronous execution of the actions of the different loaders represents a further challenge. In the SARL approach, this means that not every accumulated experience contains a subsequent state where the previously selected action has been fully executed or completed. Furthermore, the reward is also delivered late, when the request in the subsequent time step is triggered by another loader. This contradiction to the theory of a Markov Decision Process (MDP) hinders the success and speed of finding the optimal policy.

This thesis has demonstrated how these challenges can be addressed through the application of a Multi-Agent Reinforcement Learning (MARL) approach. The key improvements include:

- **Selective State Distribution**: By limiting the distribution of the system state only to the responsible agent, it is ensured that the agent's previous action is fully executed in the subsequent state.
- **Partial Observability**: The partial observation of the agents effectively reduces the size of the local state and action spaces without compromising the efficacy of the learned joint strategy.
- **Work Area Restriction**: Limiting the work areas of the loaders has been shown to be a legitimate restriction, reducing the system complexity and state space, as this division is learned independently by the agents.
- **Reward Function Adjustments**: Although graduated rewards to prioritize rapid transfer did not yield notable enhancements, modifying the reward function successfully prevented consecutive drives caused by the simplified simulation.

The operational efficacy of the proposed methodology has been validated through several iterative experiments. The findings indicate that multiple agents can independently

learn and yet still collaborate, thus identifying a cooperative strategy for maximizing throughput in the simulated system.

8.1 FUTURE PROSPECTS

In the course of preparing this master's thesis and, in particular, in the process of conducting the experiments, some minor issues emerged that would benefit from further investigation in future works. The experiments demonstrated the necessity for the development of a more effective metric for the evaluation of models during training. In addition to the throughput achieved, this metric should also take into account other values, such as the number of actions required. From a practical standpoint, it would also be beneficial to ascertain which hyperparameters influence the training and learning behavior of the agents and to what extent.

Additionally, it would be valuable to examine the performance of the MARL approach when the simulated system is more similar to a real system. Such an investigation could entail examining systems with additional loaders and stations with varying degrees of overlap between their work areas, as well as more intricate production processes involving different product types with distinct processing times in the machines. Moreover, examining the influence of the structure of the Deep Q-Network (DQN) on the training process may prove advantageous in scenarios where the complexity of the production plant and, consequently, the size of the state spaces increases.

Furthermore, deficiencies in the simulation software that impede the effectiveness of RL have been identified. Potential areas for future investigation include the removal of the penalty for double runs in instances where the simulation accurately maps the acceleration of the loaders. Additionally, it is hypothesized that the learning behavior can be enhanced by mapping the restricted work areas and regulating the requests to the agents in a manner that aligns with the simulation's representation of the physical system.

Part II

APPENDIX

A

TABLES

Table A.1: Detailed overview of number of potential values for each feature in the state space cardinality, considering the different approaches and responsibilities. In the event that a value is not reachable, such as the position *Conveyor_In* for loader 2, the number of potential values is reduced for that feature. The station state only has four valid combinations even though the station state itself is represented by three boolean values.

		Loader 1				Loader 2					Station						
	Requesting Loader	Position	Gripper	Action	Blocked	Position	Gripper	Action	Blocked	Conveyor_In	A	B	C	D	E	F	**State Space Cardinality**
SARL Single I-Loader																	
	-	8	5	-	-	-	-	-	-	2	4	4	4	4	4	4	327 680
SARL Two I-Loaders																	
	2	7	5	11	2	7	4	11	2	2	4	4	4	4	4	4	7 771 258 880
MARL Responsibility for All Stations																	
Agent 1	-	7	5	-	-	7	-	11	2	2	4	4	4	4	4	4	44 154 880
Agent 2	-	7	-	11	2	7	4	-	-	-	4	4	4	4	4	4	17 661 952
MARL Restricted Work Area																	
Agent 1	-	5	4	-	-	5	-	9	2	2	4	4	4	4	-	-	921 600
Agent 2	-	5	-	9	2	5	3	-	-	-	-	-	4	4	4	4	345 600

B

FIGURES

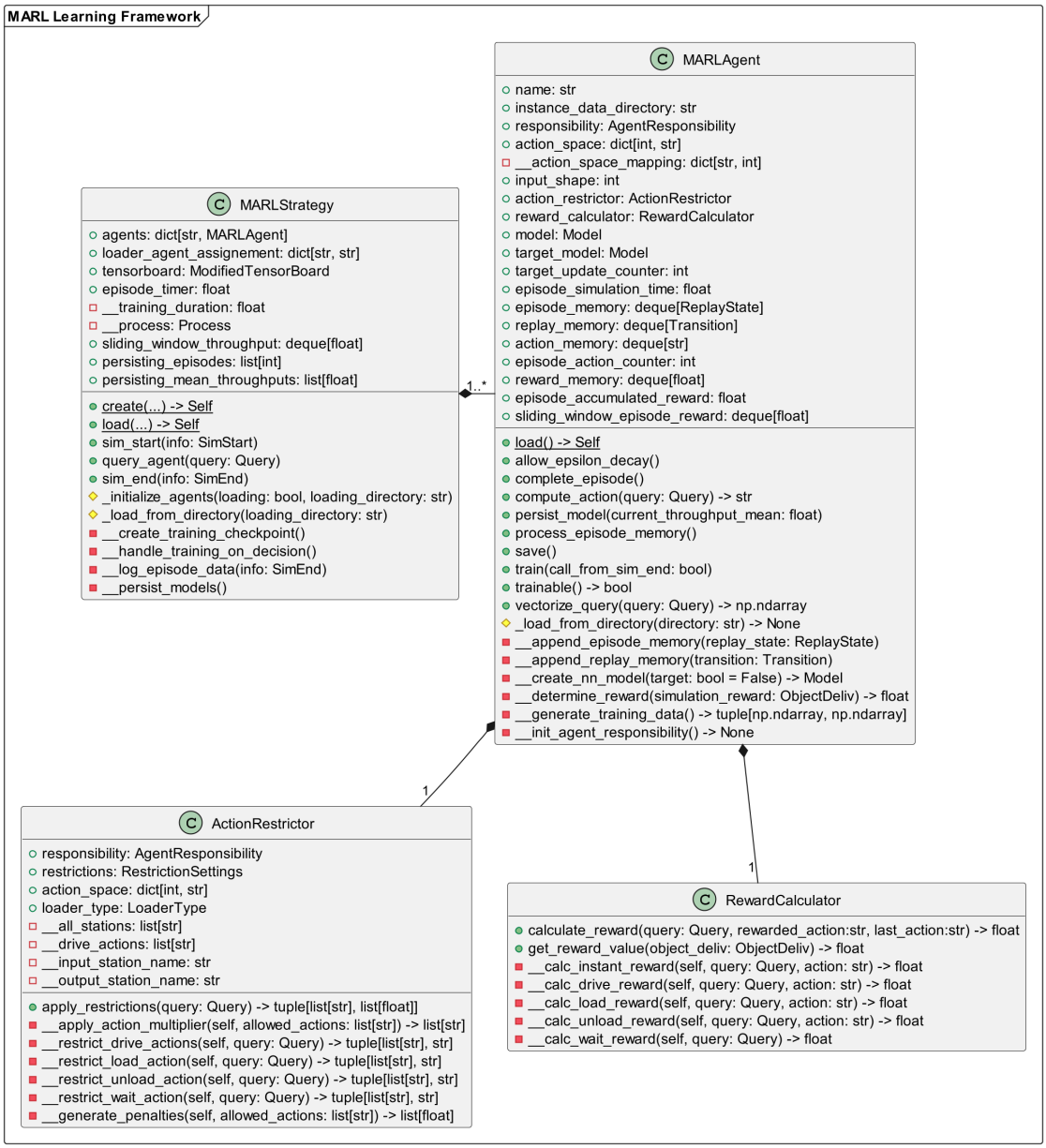

Figure B.1: Detailed class diagram with methods and attributes for the MARL approach.

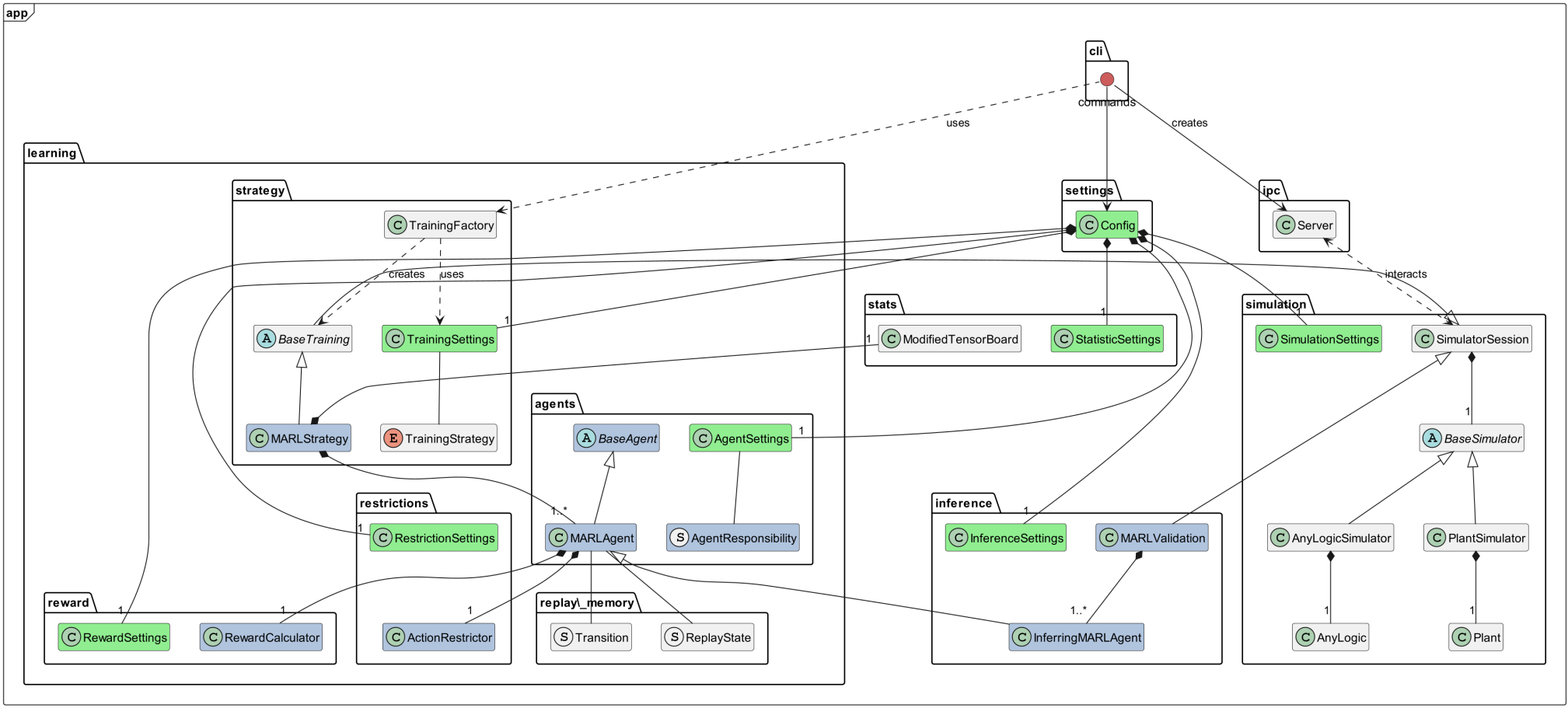

Figure B.2: Overview of all packages and their classes employed in the MARL approach with their relations. The classes implemented during this thesis are marked in light blue. The settings classes for the different modules are marked in green.
Some classes have slightly alternative names in the code, which are presented here in simplified form.

BIBLIOGRAPHY

[Aba+15] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[ACS24] Stefano V. Albrecht, Filippos Christianos, and Lukas Schäfer. *Multi-Agent Reinforcement Learning: Foundations and Modern Approaches*. MIT Press, 2024. URL: https://www.marl-book.com.

[BBC57] R. Bellman, R.E. Bellman, and Rand Corporation. *Dynamic Programming*. Rand Corporation research study. Princeton University Press, 1957. URL: https://books.google.de/books?id=rZW4ugAACAAJ.

[Bis06] Christopher M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Berlin, Heidelberg: Springer-Verlag, 2006. ISBN: 0387310738.

[DS13] Ke-Lin Du and M.N.s Swamy. *Neural Networks and Statistical Learning*. Oct. 2013. ISBN: 978-1-4471-5570-6. DOI: 10.1007/978-1-4471-5571-3.

[GD22] Sven Gronauer and Klaus Diepold. "Multi-agent deep reinforcement learning: a survey." In: *Artificial Intelligence Review* 55.2 (2022), pp. 895–943. ISSN: 1573-7462. DOI: 10.1007/s10462-021-09996-w. URL: https://doi.org/10.1007/s10462-021-09996-w.

[GEK17] Jayesh K. Gupta, Maxim Egorov, and Mykel Kochenderfer. "Cooperative Multi-agent Control Using Deep Reinforcement Learning." In: *Autonomous Agents and Multiagent Systems*. Ed. by Gita Sukthankar and Juan A. Rodriguez-Aguilar. Cham: Springer International Publishing, 2017, pp. 66–83. ISBN: 978-3-319-71682-4.

[HSW89] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators." In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[Krn+24] Aleksandar Krnjaic et al. *Scalable Multi-Agent Reinforcement Learning for Warehouse Logistics with Robotic and Human Co-Workers*. 2024. arXiv: 2212.11498 [cs.LG]. URL: https://arxiv.org/abs/2212.11498.

[Lin92] Long-Ji Lin. "Self-Improving Reactive Agents Based on Reinforcement Learning, Planning and Teaching." In: *Mach. Learn.* 8.3–4 (1992), 293–321. ISSN: 0885-6125. DOI: 10.1007/BF00992699. URL: https://doi.org/10.1007/BF00992699.

[Mil+23] Robert Miltenberger et al. "Integration von Simulation und Reinforcement Learning zur Portalrobotersteuerung." de. In: *Simulation in Produktion und Logistik 2023*. Ed. by Sören Bergmann et al. Ilmenau: Universitätsverlag Ilmenau, 2023, pp. 145–154. ISBN: 978-3-86360-276-5. DOI: 10.22032/dbt.57786. URL: https://doi.org/10.22032/dbt.57786.

[Mni+15] Volodymyr Mnih et al. "Human-level control through deep reinforcement learning." In: *Nature* 518.7540 (2015), pp. 529–533.

[Pla] PlantUML Team. Paris, France: PlantUML. URL: https://plantuml.com/en/.

[Pyt] Python Core Team. *Python: A dynamic, open source programming language*. Python version 3.9. Python Software Foundation. URL: https://www.python.org/.

[RHW86] David E. Rumelhart, Geoffrey E. Hinton, and Ronald J. Williams. "Learning representations by back-propagating errors." In: *Nature* 323.6088 (1986), pp. 533–536. ISSN: 1476-4687. DOI: 10.1038/323533a0. URL: https://doi.org/10.1038/323533a0.

[SSSS16] Shai Shalev-Shwartz, Shaked Shammah, and Amnon Shashua. *Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving*. 2016. arXiv: 1610.03295 [cs.AI]. URL: https://arxiv.org/abs/1610.03295.

[Sil+18] David Silver et al. "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play." In: *Science* 362.6419 (2018), pp. 1140–1144. DOI: 10.1126/science.aar6404. eprint: https://www.science.org/doi/pdf/10.1126/science.aar6404. URL: https://www.science.org/doi/abs/10.1126/science.aar6404.

[SB18] Richard S. Sutton and Andrew G. Barto. *Reinforcement Learning: An Introduction*. Second. The MIT Press, 2018.

[Tam+15] Ardi Tampuu et al. *Multiagent Cooperation and Competition with Deep Reinforcement Learning*. 2015. arXiv: 1511.08779 [cs.AI]. URL: https://arxiv.org/abs/1511.08779.

[Vin+19] Oriol Vinyals et al. "Grandmaster level in StarCraft II using multi-agent reinforcement learning." In: *Nature* 575.7782 (2019), pp. 350–354. ISSN: 1476-4687. DOI: 10.1038/s41586-019-1724-z. URL: https://doi.org/10.1038/s41586-019-1724-z.

[WD92] Christopher J. C. H. Watkins and Peter Dayan. "Q-learning." In: *Machine Learning* 8.3 (1992), pp. 279–292. ISSN: 1573-0565. DOI: 10.1007/BF00992698. URL: https://doi.org/10.1007/BF00992698.

[Yu+23] Chao Yu et al. *Asynchronous Multi-Agent Reinforcement Learning for Efficient Real-Time Multi-Robot Cooperative Exploration*. 2023. arXiv: 2301.03398 [cs.RO]. URL: https://arxiv.org/abs/2301.03398.

[Zho+22] Wei Zhou et al. "Multi-agent reinforcement learning for cooperative lane changing of connected and autonomous vehicles in mixed traffic." In: *Autonomous Intelligent Systems* 2.1 (2022), p. 5. ISSN: 2730-616X. DOI: 10.1007/s43684-022-00023-5. URL: https://doi.org/10.1007/s43684-022-00023-5.

[Zis+24] Horst Zisgen et al. "Dynamic Scheduling of Gantry Robots using Simulation and Reinforcement Learning." In: *Proceedings of the Winter Simulation Conference*. WSC '23. San Antonio, Texas, USA: IEEE Press, 2024, 3026–3034. ISBN: 9798350369663.