
Master's degree

Development engineer mechanical engineering

---

# Tetromino-based process optimization of robot-assisted charging and Tessellation tasks using neural nets

Submitted as a master's thesis

for obtaining the academic degree

..

Graduate engineer for technical-scientific professions (Dipl.-Ing.)

from

Christian Brandstätter, BSc.

Aug 2021

---

supervision of the work

FH Prof. dr Roman Froschauer

# Foreword/Acknowledgments

I would like to thank my supervisor Mr. FH-Prof. DI (FH) Dr. technical novel
Franz Froschauer for the friendly support with my diploma thesis. Of
I would also like to thank the employees from the Rubig company for the good cooperation.

Special thanks go to my family and friends for their support w ah
during my studies. Thanks.

# Statutory declaration

I declare in lieu of an oath that I have written this work independently and without outside help, and that I have not used any sources other than those specified and that passages taken from the sources used as such have made known.

The work has so far been in the same or a similar form not submitted to any other testing authority and also not published.

The present work is with the electronically over- averaged text document identical.

. . . . . .

Christian Brandstätter, BSc.
Catfish, August 2021

# short version

In this work, an automation solution scheme for the charging process of hardening plants is developed. The charging includes the storage of components on a charging frame and equipping the hardening systems with these frames.
At the beginning of the work there is a specification of the task the process to be mapped is examined and placed in the context of the task description posed.
In the course of processing, the current state of the art is developed and backed up with literature. The research focuses on the areas of automation structure, communication, software development, tessellation and artificial Intelligence. Concepts and their contents are created on the basis of this knowledge articulated. Then the contents of the concepts are implemented for a prototype.

A configuration program is created that defines the participating system components and enables them to be arranged geometrically.
The generated layout is then created in a 2D simulation environment loaded and animated using a previously defined process logic, which includes the tiling of the components and the course of the overall process. Furthermore, this program is used to establish a connection to a robot training cell and thus to transfer the process logic to a real system wear.
For the tiling of complex components, an approximation by Tetrominos is carried out. The tessellation problem that arises in this way is solved in a first approach using neural networks.
Solution approaches and ideas for further content of an implementation follow, such as expansion of the functionality and suggestions for improving the implementation. Included Possibilities are also mentioned that could not be included in the work due to time constraints.

# Abstracts

In this work, an automation solution scheme for the charging process of hardening plants is developed. Charging includes the depositing of components on a charging rack and the loading of the hardening systems with these racks. At the beginning a concretization of the given task is taking place. Furthermore, the process to be mapped is examined under the context of the task description.
In the course of this, the current state of the art is elaborated and supported with proper literature. The research focuses on the areas of automation structure, communication, software development, tiling and artificial intelligence. Based on this knowledge, concepts are created and their contents structured. Therefore, the contents of concepts are implemented for a prototype.
A configuration program is created that defines the participating system components and enables them to be arranged geometrically. Then, the created layout is loaded into a created 2D simulation environment and is animated using a predefined process logic. The process logic includes the tiling of the components and sequences of the overall process. This program is also used to establish a connection to a robot training cell and hence, transfers the process logic to a real system.
For packing of complex components, an approximation through Tetriminoes is car ried out. The resulting packing problem is solved in a first approach with the help of neural networks.
This is followed by solutions, approaches and ideas for further contents of an implementation, such as the extension of functionality and suggestions for application improvements. Possibilities are also mentioned that could not be included in the work due to time constraints.

# table of contents

# 1 Introduction

In this chapter, the problem is presented and suitable research questions are derived from it. Then goals and non-goals are defined and a appropriate guideline selected for concept development.

## 1.1 Motivation

In cooperation with the Rubig company, a suitable software architecture and approach is being sought in order to advance the automation of their systems. The actual desire is to automate the placement of plasma nitriding systems as far as possible and to market this as an automation solution. The systems are process systems into which components are fed to harden their surface. The charging of these systems happens currently mostly by hand. In the future, this should be done by robots and assembly lines be led.

# 1.2 Issue

The initial problem consists in finding suitable data models and communication protocols in order to network the desired system components (robot, controller, etc.) and to select their parameters as automatically as possible depending on the component. Subsequently, based on this, automation processes for the charging of plasma nitriding systems are to be derived.
The focus is on the tiling of the system and the necessary for the integration software architecture.

## 1.3 Research question

The following questions can be derived from the presented problem:

– What could an interaction scheme for charging plasma nitriding systems look like and how can such a system be designed to be as reusable as possible?

– How could data models for the algorithmically optimized charging of the plant look like?

## 1.4 Aim and content of this work

The main focus of this work lies in the creation of a system for fully-autonomous tiling of the charging levels. For this purpose, among other things, the technology of artificial intelligence (AI) is used and tessellation is treated as an optimization problem. Furthermore, a possibility is developed, this technology involve in a process. Specifically, the following goals and non-goals can be defined be ned:

Table 1.1: Goals and non-goals

| goals | non-target |
|---|---|
| Develop process configuration | full configurability |
| Charging of component classes | Classification of components into classes |
| Develop process control scheme | Mapping of all process flows |
| Develop interaction scheme for users | user analytics |
| Integration into a mobile training cell | Integration in test systems |

## 1.5 Implementation Strategy

In order to find a suitable area for processing the topic content, the work is divided into four categories. At the beginning of this chapter, the requirements were specified and research questions as well as goals and non-goals were defined.
A technical research is then carried out in the following chapter in order to disclose the state of the art and possibilities. The information obtained is then summarized in a concept development.

Finally, the presented concept is implemented as a prototype for validation, which takes into account the previously discussed concept features. The structure of the work is largely based on the V model (Fig. 1.1) [1], which is based on the Verein German Engineers (VDI) guideline 2206 oriented. At the beginning, as in the Figure shown that defines project requirements. Subsequently dedicated

one looks at the basic determination, with the help of which the system design is built and a prototype is created.

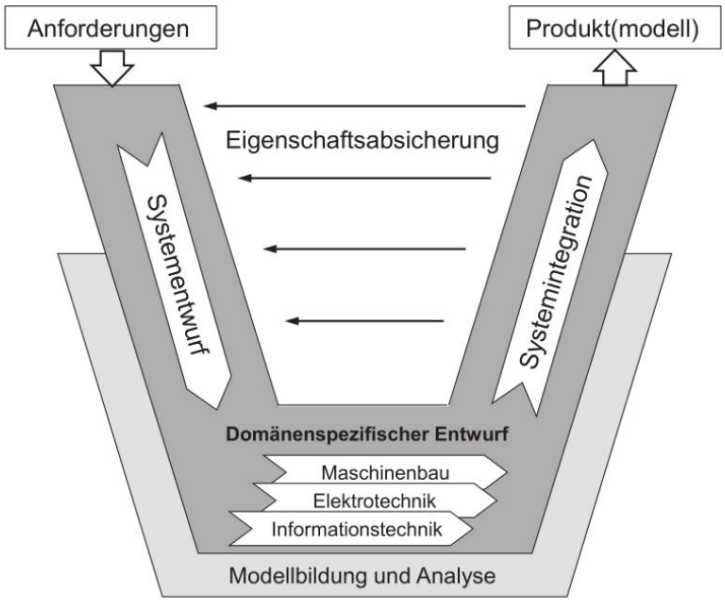



Figure 1.1: V model [1]

## 1.6 Requirements

Care was taken to treat the content briefly but comprehensively,
give the reader a good overview, not everything can flow into the work and for this reason to convey the facts. Nevertheless, in order to basic knowledge is omitted
in the following areas:

- Programming: syntax in Python.
- Automation: Tasks of a control-programmable control (PLC).
- Mathematics: matrix and vector calculation, statistical and probable theory, algebra, differential equations and numerics.

# 2 research

In this chapter, research on the existing problem is carried out.
It starts with the classic automation pyramid, followed by a
Standard for programming controllers: International Electrotechnical Commission (IEC)
61131-3. Subsequently, after viewing the control page
a protocol for communicating with these controllers is presented, namely Open Platform
Communication Unified Architecture (OPC UA). After that
a modeling language for the description of software architectures is presented, which
is used to schematize one's own program routines. Subsequently, the geometric tiling is
presented, from which concepts for
be derived from the charging process. Last but not least is the area
Machine Learning (ML) processed, whereby a large part is used for the realization of the
own idea.

## 2.1 Automation Pyramid

The automation pyramid [2] (Fig. 2.1) was developed to illustrate industrial automation. It shows the essential levels in one
Companies again, these differ in terms of speed and the
Amount of data to be transferred. A distinction is made between the following hierarchical levels:

- Company management level: This level serves to manage the company with regard to marketing, personnel, product and investment planning.
- Plant management level: In this level, production planning, scheduling monitoring and cost analysis. In addition, the administration and processing of delivery orders carried out here.
- Production management level: This deals with short-term production planning, such as the scheduling of machines and personnel.
- Process control level: Also called cell level, the process control level is used for Control or regulation of production processes. The term cell level comes from the fact that the processes often take place in separate production areas fen.
- Field level: This is where data is collected, often in real time, and according to a instruction responds. The data is also processed at this level and made available to the higher-level system.

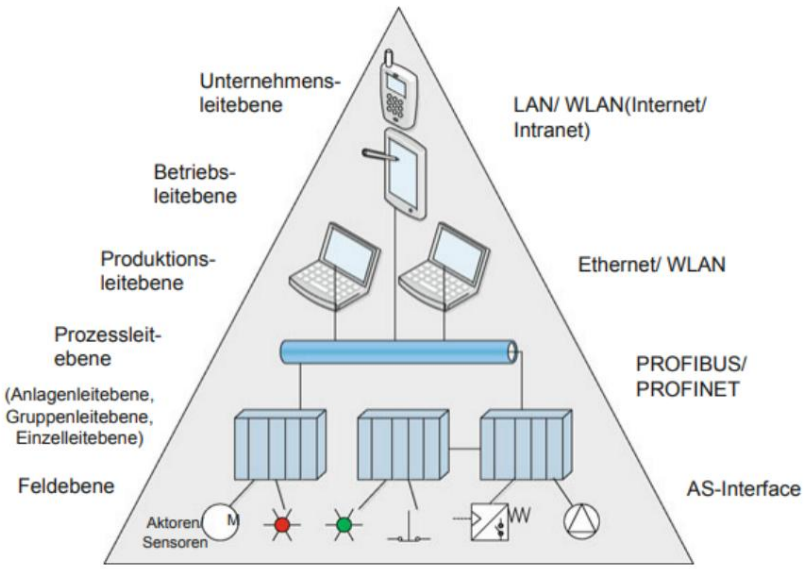



Figure 2.1: Automation pyramid [2]

## 2.2IEC 61131-3

In industry, automation solutions usually contain PLCs that are responsible for processing control logic. This also includes not only the processing of the logic, but also the addressing of relays and the transmission of data. Since the controller manufacturers introduced different programming languages in the beginning, the cry for a standard grew, um to fix the resulting incompatibility problems. This default could be introduced with the standard IEC 61131-3 [3], which meanwhile covers all controller manufacturers have usually fully implemented in their controllers.

### 2.2.1 Project structure

In order to standardize the structure of PLC projects, the IEC introduced a hierarchy under standard number 61131-3. The hierarchy builds up over Configuration elements and the program organization unit (POU). Alone The elements shown in Fig. 2.2 can contain several sub-elements.

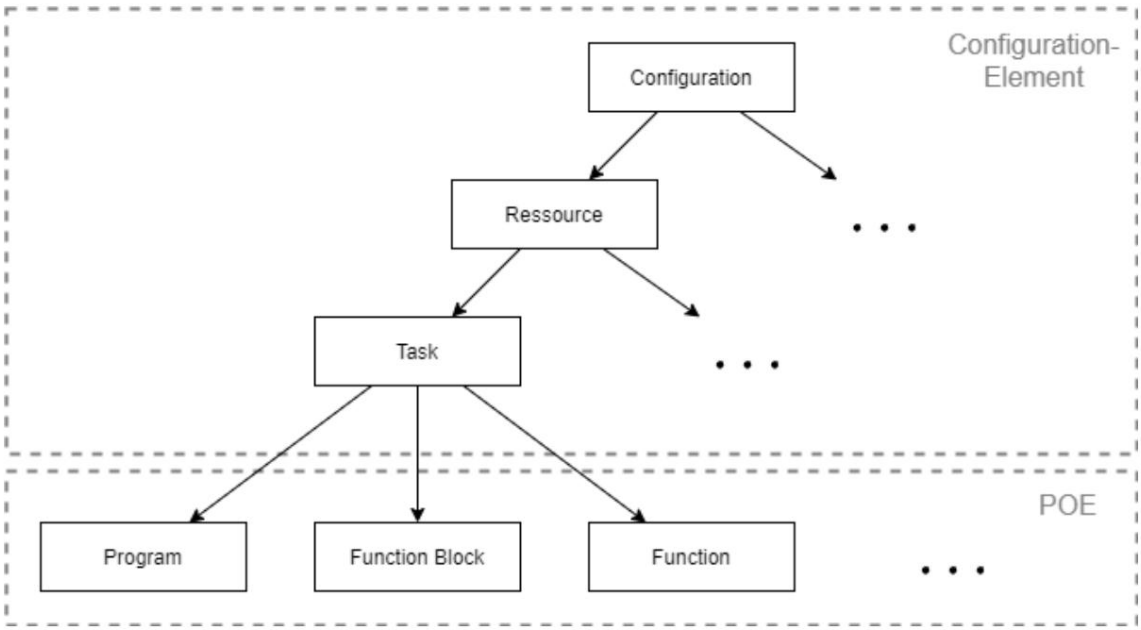



Figure 2.2: Project structure according to IEC 61131-3

### 2.2.2 Configuration Items

The configuration items are configuration, resource and task as shown in Fig. 2.2. These elements do not contain any code themselves, but serve
only the configuration of the controller. There is usually a ¨ for this
Settings page in the manufacturer's development environment.

- Configuration: The configuration defines the available resources such as e.g. B. bus modules and I/O cards, but also global variables
  can be created here.
- Resource: They stand for the processing units of the SPS of the Central Processing Unit (CPU). Variables can again be declared for the individual resources
  that are available in the underlying components.
- Task: A task defines the runtime properties of programs. A program written in different programming languages can be called up and run through cyclically or when certain events are fulfilled. This and the assignment of a priority for program execution
  can be declared in a task.

### 2.2.3 Program Organization Unit

The POU maps the actual logic of the PLC program. Variables can again be created in a POU, but these can now only be called up within the program. In addition to the variables, the user can also call up programs and function blocks if they have been created in a library. It should be noted here that function blocks cannot be called by functions, only by other, upstream function blocks. However, they can call functions themselves. In contrast to a function block, a function can only return one variable and does not have its own memory area. The memory area allows a function block to store variables, giving the function block the ability to remember states.

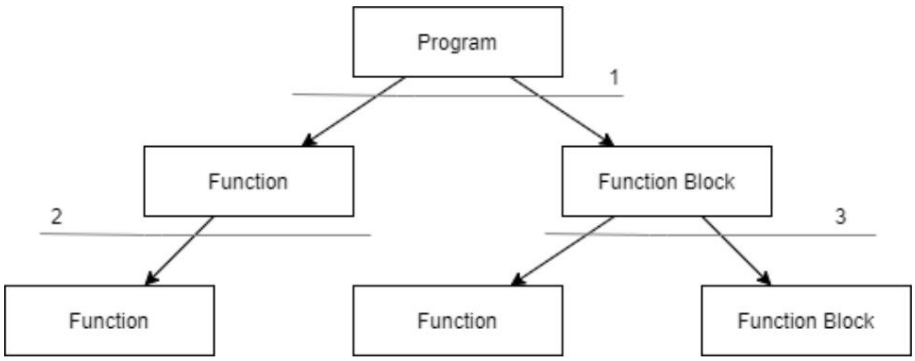



Figure 2.3: POU structure

1: The program calls a function or function block.

2: Function calls function.

3: Function block calls function or function block.

### 2.2.4 Variables and Data Types

IEC 61131-3 defines common standard data types such as boolean, byte or integer. However, you can also define and use your own data types, such as:

– Structure types

– Enumeration types

– alias types

A data type can be defined within the configuration or within
be done in a library. Figure 2.4 shows the definition of the data types within the configuration of the Codesys programming environment.

```
TYPE TestStructure :
STRUCT
    a:INT;
    b:BOOL;
    c:REAL;
END_STRUCT
END_TYPE
```

(a) Structure

```
TYPE TestEnumeration :
(
    member1 := 0,
    member2 := 5,
    member3 := 10
);
END_TYPE
```

(b) Enumeration

```
TYPE TestAlias : INT;
END_TYPE
```

(c) alias

Figure 2.4: Definition of the data types

Variables are defined either in the configuration, in the resource definition
or defined within a POU. In the definition itself, in addition to Va
an initial value can also be specified for the variable name and the variable type. The Access behavior of a variable depends on the location in which the variable
was created.

## 2.2.5 Programming Languages

The IEC 61131-3 defines 5 programming languages. These can be used
to map the required logic. Some of these are more abstract than others.
they offer the possibility to map program routines of different levels of complexity simply or less simply. To provide a brief insight,
four of the programming languages explained using a motor control. The controller switches a motor relay if a button is pressed and the limit switch
is not reached. A switch is also used to check whether the relay has already been switched (Fig. 2.5).

– Ladder diagram: No programming knowledge is required to read this program code, since it represents the logic purely graphically. The structure is reminiscent of a flow chart from electrical engineering. The programming objects
are simple control technology components such as relays and limit switches.
These are connected in series or in parallel, using the program code of
processed left to right. In the example, the motor relay
switched if the button 'button' is pressed or the switch '%IX23.5'
is already active. In addition, the closer must be active, i.e. the variable
'EmStop', deliver a zero signal.

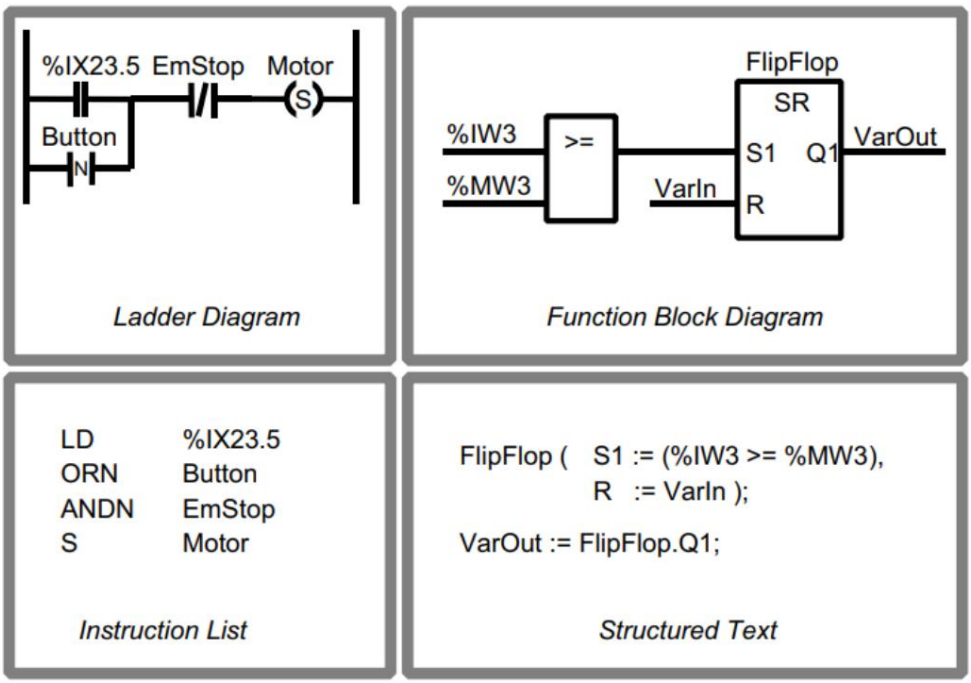



Figure 2.5: Programming languages of IEC 61131-3 [3]

– Function block language: As the name suggests, it consists of
function blocks. A function block usually maps real objects. Similar to classes in object-oriented programming, the function block attempts to accurately represent the properties of objects. function blocks
have their own memory area and can therefore be used by previous ones events are affected. The engine control is implemented with two blocks, first the threshold value check using a greater than or equal block
and then switching the relay using a flip-flop. The flip-flop remembers when a pulse arrives at input 'S1' and switches accordingly
for a long time until a pulse is also effective at input 'R'. As a note
mentioned that function blocks themselves can be constructed with different programming languages.

– Instruction List: Is a very low-level programming language and ori
refers to assembler. Showing variables from memory
loaded into the cache memory and then performed operations on it. Once all the desired operations have been carried out, the current value in the cache is transferred back to the main memory overwritten. For example

first the variable '%IX23.5' is loaded into the cache. Then will
an 'or' and 'and' operation on the cache with the variables 'button' and

'EmStop' carried out. The result of these operations is transferred to the 'Motor' variable.

– Structured text: Probably the most versatile programming language of the IEC standard. It is strongly based on the programming language C or Pascal and it is therefore still possible to represent complex logic in a relatively clear manner, at least for users who are used to dealing with high-level languages. In of this programming language, function blocks are often used to make code more readable. In the example, an object of the data type 'FlipFlop' created and initialized with the values of ‚%IW3', ‚%MW3' and ‚VarIn'. The condition for exceeding the threshold value can be written particularly compactly with the greater than or equal to operator, which then the internal variable 'S1' of the flip-flop.

– Sequence language: This language consists of steps and transitions. The steps contain the program flow, which can be implemented using other programming languages. If a transition condition is fulfilled, is switched to the respective step and the underlying program sequence is carried out.

## 2.3 OPC Unified Architecture

OPC UA [4] [5] [6] is a service-oriented software architecture. The standardized and open structure enables data to be exchanged independently of the PLC manufacturer. In most cases, however, compatibility must be activated for a fee. OPC UA mainly deals with data modeling and the data transport mechanism. Data is exchanged using an adapted standard protocol from network technology, the Transmission Control Protocol (TCP), which defines the type of data exchange. UATCP is

Suitable for fast binary data exchange, for encrypted data exchange

OPC UA uses the XML format, which is transmitted via Hypertext Transfer Protocol Secure (HTTPS) protocols (Section 2.3.6). In order to set up such a data connection, the server needs a valid network address and a port, which make up a Uniform Resource Locator (URL).

### 2.3.1 Address Space

The data exchange takes place with so-called nodes, which serve as data models
are understand. Furthermore, nodes can also be interpreted as an address block in memory,
which contains node attributes, data and references to other nodes
contain. An important part of the node attributes is the node identifier (ID)
It consists of 3 components that enable localization in memory. The three components are:
namespace-Uniform Resource Identifier (URI),
Data type of the identifier and the identifier itself. Here is a short example:

Program 2.1: OPC UA Node ID

```
1 ns= http : // example . org:80/ some / directory ; string = Temp
```

However, the node ID can be shortened even further, namely by introducing a lookup table
(Tab. 2.1). Thus the namespace is reduced to one
integer value.

Table 2.1: Abbreviation of the namespace URI using a lookup table

| index | Namespace URI |
|---|---|
| 0 | http://opcfoundation.org:80/UA |
| 1 | http://example.org:80/some/directory |
| ... | ... |

### 2.3.2 Browsing

The client has the option of searching through the address space
Make a browsing request to the OPC UA server. In doing so, the server is given a
output-node/ start-node, which in turn transmits all references of this node
returns. The process is illustrated graphically in Fig. 2.6, the green nodes are returned un mar-
from the server and the arrows represent the
respective references.

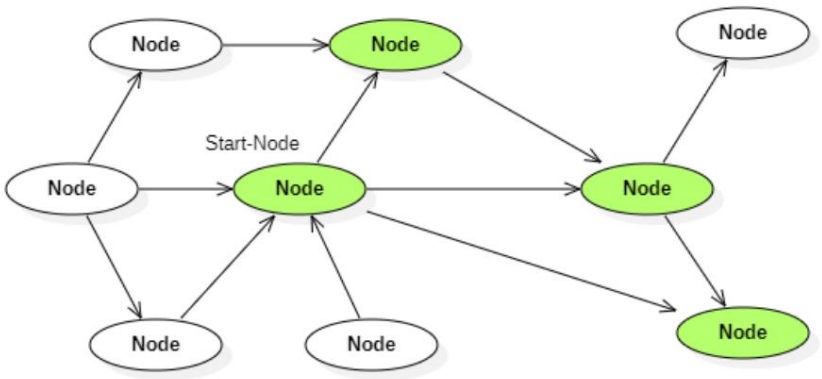



Figure 2.6: Browsing in the address space

In addition, there is the possibility of using views, with the help of which the address space can be divided into subspaces. Will now submit a search sent to the server, only those references of the source node are returned, ¨ located in this subspace (Fig. 2.7).

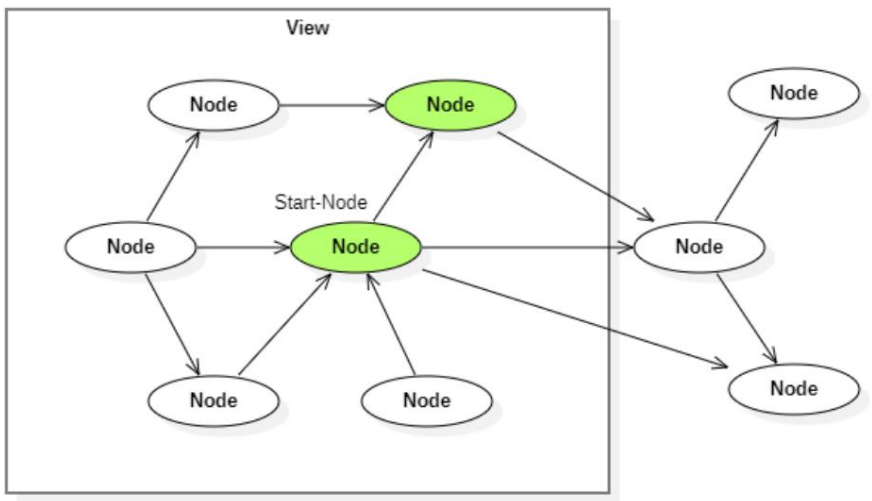



Figure 2.7 Browsing the address space with a view

### 2.3.3 Subscription

Should the client notice value changes without calling up the value cyclically need to, OPC UA offers the possibility to create a subscription. The subscription consists of one or more monitored items that have a reference to the have a monitoring node attribute. In addition, a buffer size and a

Sampling interval can be set. If the value changes, the client is notified by the server. A change in value is recognized on the server side by means of a filter that determines when a value is to be considered changed. In order to receive a notification from the server at certain times, a publish interval can be set. This describes the minimum time difference that may elapse between two messages. This can be used to to prove the functionality of the subscription.

### 2.3.4 Node Model

In order to set up the data models in a standardized way, the OPC UA Foundation designed a node model (2.8). This model states that all nodes of derived from a base-node-class that has all the basic attributes of a node own. These basic attributes are: 'NodeID', 'NodeClass', 'BrowsingName' and 'DisplayName'. The model also provides for the use of some standard node types like 'Object', 'Variable' and 'Method'.

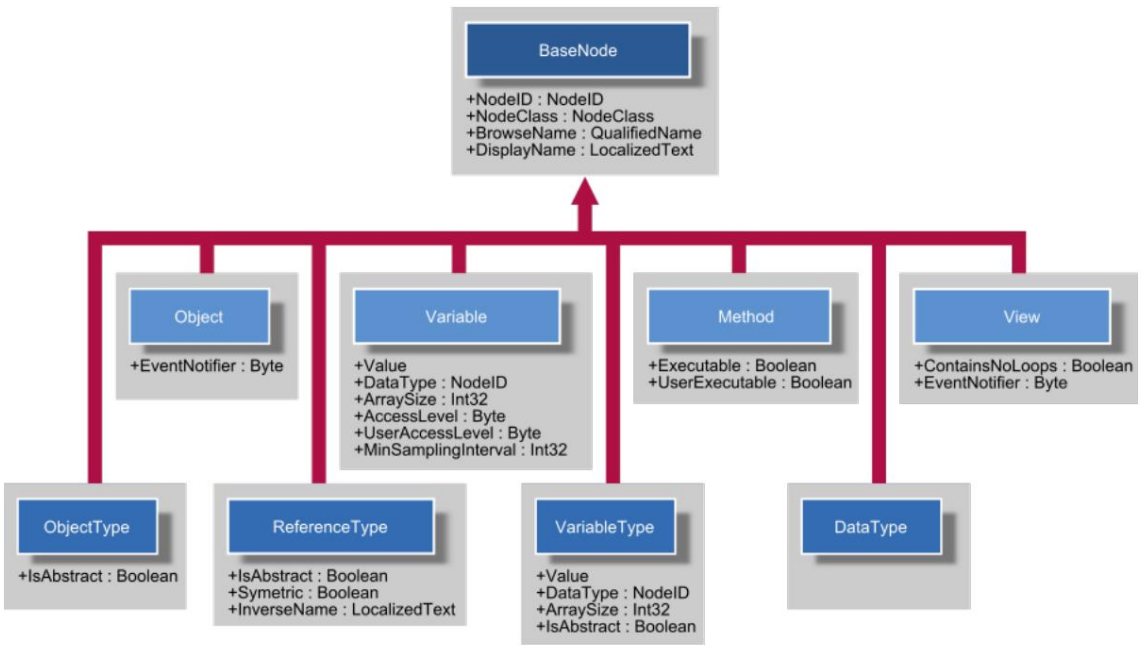



Figure 2.8: Node model of OPC UA [5]

### 2.3.5 Node type

If the nodes are defined and implemented, one speaks of an OPC UA basic information model. The model contains all the information needed to to map the process. In order to interact with the model, the

The nodes made available to the server are grouped together. For this Grouping, the following declarations are provided by OPC UA:

- OPC DA (Data Access): Real-time reading of variables for functions such as read(), write(), browse() or subscribe()
- OPC HA (Historical Access): Reading of already stored data, for radio functions like analyze(), collect()
- OPC A&E (Alarm & Events): notifications for alarms and events, for functions like receive(), filter()
- OPC Prog (Programs): Executes a saved procedure

### 2.3.6 Communication Channels

As mentioned at the beginning, OPC UA relies on the TCP protocol for communication. Figure 2.9 shows the different communication paths, which in turn are based on already established protocols such as Hypertext Transfer Protocol (HTTP) and HTTPS, set up. The left branch 'UA Binary' is considered particularly secure and efficiently advertised [4].

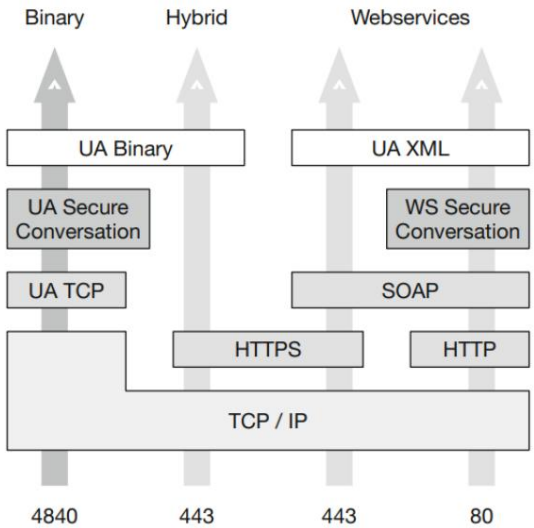



Figure 2.9: OPC UA protocols and possible communication paths [4]

When used correctly, OPC UA offers a high level of security compared to other industrial protocols [7]. In addition to the classic username/password combination, other authentication methods such as web tokens and certificates are also available. The encryption methods used methods are:

- Basic-128/RSA-15
- Basic 256
- Basic-256/SHA-256
- AES-128/SHA-256 (RSA-OAEP)
- AES-256/SHA-256 (RSA PSS)

### 2.3.7 Function and program call

Methods are used to make function calls on the server.
The client receives information about the transfer and return parameters
meter and their data types. If the client makes a request to execute a method with the
required transfer parameters, the server works through the corresponding processes and
transmits the return para- when the process ends
meters to the client. Due to this process, methods are only suitable for short-lived
processes.

In contrast to the methods, OPC UA programs map a complete state machine, giving
the client a higher degree of control over the
Process. This gives the user the opportunity to start, stop or
to pause and is also informed by the server about the status changes, so-called
transitions. The client transfers the start method to the
required transfer parameters and then receives notifications from the server about each
transition that has taken place and any intermediate results. in the
running process, the client has the option of using other methods
to control the process flow. When the process is finished, the client can use the jerk
query output parameters. Compared to methods, programs are suitable for
longer-lived processes, as represented by state machines.

## 2.4 Unified Markup Language

Unified Markup Language (UML) is an international standardized modeling language
and is managed by an international organization, the Object
Management Group (OMG). Using UML, a software design can take place,
in order to convey the clearest possible picture of its functionality. All
Explanations of this chapter were taken from [8] and with the help of our own examples
worked out.

### 2.4.1 class

A class is used to describe objects that are involved in the functioning of the application.
A class is represented by a rectangle, which
Contains the name of the class. Below is another one of the same width
Rectangle for specifying attributes. Furthermore, another rectangle with information on
operations can be adjacent in the display. Fig. 2.10 shows this
Representation based on some examples, but without attributes or operations.


Figure 2.10: Representation of classes in UML

**object**

Objects derive from the use of classes. For example, a
Object named 'Bearing' can be instantiated by class 'Component'.
There can be any number of instances of a class.

### 2.4.2 Attribute

Attributes are characteristics of a class derived from the objects of a class
will. For example, the object 'Ball Bearing' has a mass, which is created as an attribute
of the class and assigned a value during instantiation. attributes
require a designation and a variable type. If not further specified,
the attributes are always so-called mandatory fields. Is not this
desired, this is marked by setting [0..1] after the attribute types. Furthermore, enumerations
can be used,
these are lists of all possible values that the attribute can take on. In addition to the
variable type, information for inheritance (inheritance) is also
needed. This serves to regulate access to variables after an object has been created. In
UML, this information can be represented by the following 3 characters in front of the
Display variable type:

# protected

+ public

- private

Figure 2.11 shows some of these representations for attributes. At this point it should be mentioned that Python does not provide any method for access regulation and therefore all attributes are marked as public.

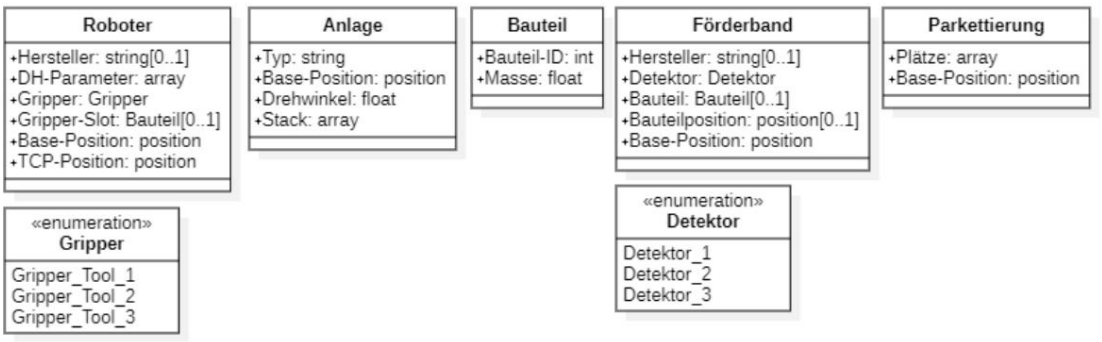



Figure 2.11: Representation of class attributes in UML

## 2.4.3 Association

The association between classes is marked with a connecting line.
Together with the attributes of a class, the association completely emulates the properties of objects.

- Multiplicity is a label of association. It specifies how many objects of one class may be connected to another. The number is given at the end of the line for the class to which it is attached. Possible multiplicities are given in Table 2.2, but you can also define your own multiplicities. One possibility would be that an object of the 'Robot' class is initialized with or without the 'Gripper' class, since a failure of the system is conceivable during a possible gripper change and the robot when the system is restarted does not necessarily have a gripper. Furthermore, the robot can normally only have one gripper mounted, unless there are several grippers on one mounting plate. In comparison, in the course of this project, an object of the class 'Plant' must also be connected to one or more objects of the class 'Tiling', since one cannot speak of a real plant if there is no tiling (Fig. 2.12).

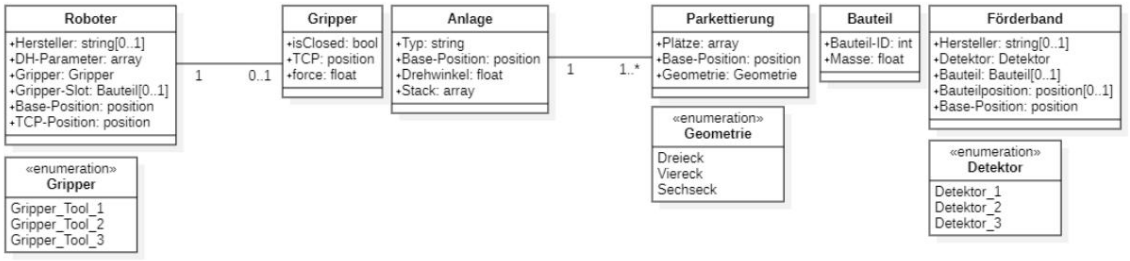



Figure 2.12: Representation of associations in UML

– An intermediate class contains additional information about the connection between two classes. For example, an intermediate class 'stack' could be used to print out the information on how many tilings of the same type a system accommodates (Fig. 2.13).

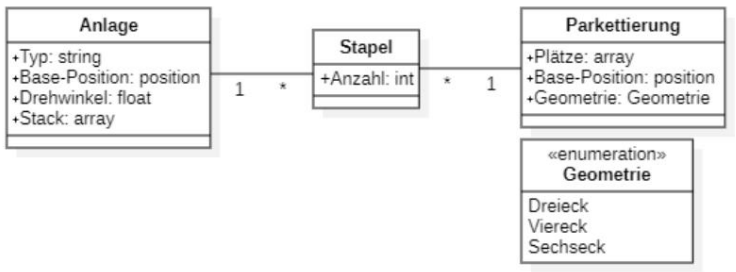



Figure 2.13: Representation of an intermediate class in UML

– Composition is used in a part-whole association. On the one hand it states that the multiplicity on the side of the whole is one and on the other hand that deleting the 'whole' side also deletes the side connected by the composition. The composition thus characterizes a deletion rule (Fig. 2.14).

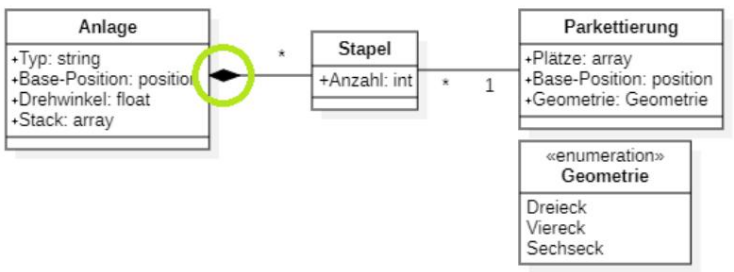



Figure 2.14: Representation of a composition in UML

– Aggregation is interesting for deletion routines and is shown as unfilled Rhombus shown (Fig. 2.15). It offers the possibility of deleting connected classes to start an interaction with the user again. In this way, the user could be given the opportunity, in addition to the 'whole' also to delete the 'part', or nothing. So you could use the aggregation as Interpret special case of composition.

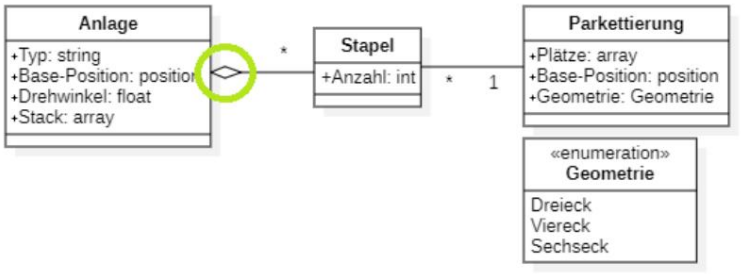



Figure 2.15: Representation of an aggregation in UML

– Navigation serves to illustrate the navigability between classes. As an example, one could consider the navigability of the previously used Illustrate the intermediate class 'stack' to the class 'tiling' as shown in Fig. 2.16. In this example, the relationship is launched alone navigable to tiling. That is, when the user views a stack, the affected tessellation is visible. However, this is the other way around not the case.

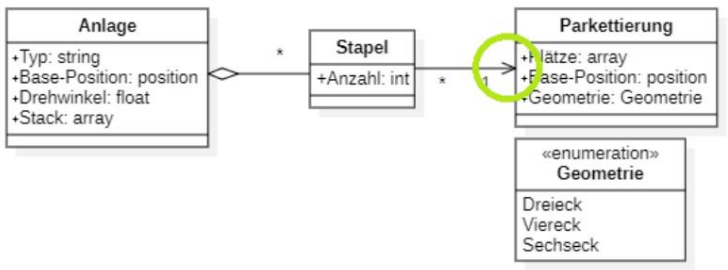



Figure 2.16: Representation of a navigation in UML

– Association name is required if the meaning of the association is not is self-explanatory. For example, an association from the class Transport posting to the class 'Location' (Fig. 2.17) without an association name would not be clear, since it is not clear whether it is a transport from location or is a transport to location.

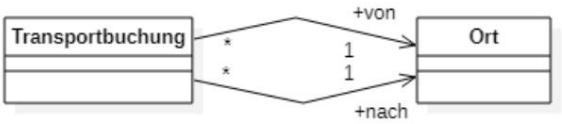



Figure 2.17: Representation of association names in UML

Table 2.2: Possibilities of multiplicities

| Multiplicity | Number of connected objects |
|---|---|
| 1 | Exactly 1 |
| 0..1 | 0 or 1 |
| * | 0 or more (abbreviation for 0..*) |
| 1..* | 1 or more |

## 2.4.4 Derived Attribute

Derived attributes are denoted by a '/' in front of the attribute name. They symbolize that the attribute is derived from other (derived) attributes
can be calculated. In the context of the project, this could mean that the class ‚tiling' derives the available spaces from the geometry information and calculated automatically (Fig. 2.18).

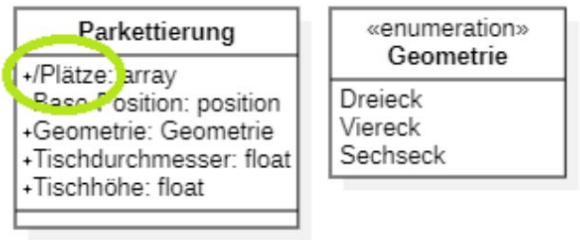



Figure 2.18: Representation of a derived attribute in UML

## 2.4.5 Inheritance

Inheritance occurs through the use of a superclass, which is also referred to as specialization. It is used when various classes have something in common
Have properties that should not be implemented again each time.
Such subclasses are derived from the superclass and inherit its attributes.
In addition, other attributes that are only derived from the
class are assigned to be defined. Inheritance in UML is through a
marked with a triangular arrow. Figure 2.19 shows the derivation of the classes
'KUKA' and 'ABB' from the 'Robot' class. The derived classes have all
Attributes of the superclass and an additional attribute about its color.

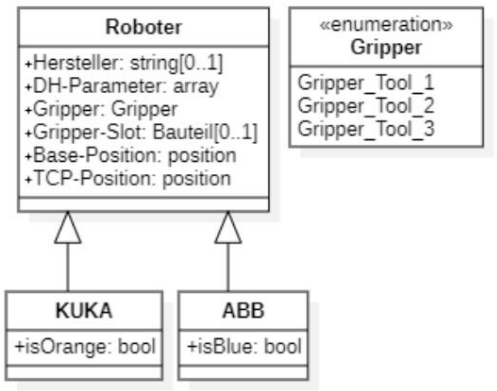



Figure 2.19: Representation of an inheritance in UML

## 2.4.6 Benefits

By using UML diagrams, a requirements specification
be reproduced analogously. This simplifies cooperation with external forces, provided they are proficient in the UML language. Furthermore
the architectures created can also be used for a database layout or a de-model of OPC UA. no

As a note, it should also be mentioned here that as a modeler it is important to weigh up whether you want to offer simplicity or flexibility. For example, for the sake of simplicity, one can choose the presented enumeration to represent the different geometries
to implement the tiling. However, if one wants to leave open the possibility
to subsequently change the properties of the geometries, reference classes had to be used be used.

### 2.4.7 Activity Diagram

The UML activity diagram describes the process that an object of a class runs through. In order to describe the process graphically, the following elements in the Activity diagram uses:

- Action: Indicates which steps are executed during the process.
- State: Indicates the status of the object running through the process, after a previous action has been performed and before the next action is performed.
- Swim Lane: Indicates by its area which class the actions are from leads.
- Start node: The start node shows the initial state that the Object accepts at initialization.
- End node: Indicates the end state of the object when the process has been completed and the state no longer changes can change.
- Decision node: Uses conditions to decide which state this is object next.
- Manual Choice: Allows the user to manually choose which Zu stood the object accepts.
- Timer: Prints out when an object assumes a certain state.
- Parallel rivers : Allows multiple states to be applied to the object at the same time and carry out different actions at the same time.
- Sub-process: A main process can execute a sub-process.
- Send signal: send a signal.
- Accept Signal: Can accept a signal sent out.

Figure 2.20 shows another simple example. The component is located after the instantiation on the conveyor belt, which is detected in the course of the process by an object of the class 'conveyor belt'. This action is in the corresponding swimming lane. Subsequently, further actions of the respective instances until the end node is reached.

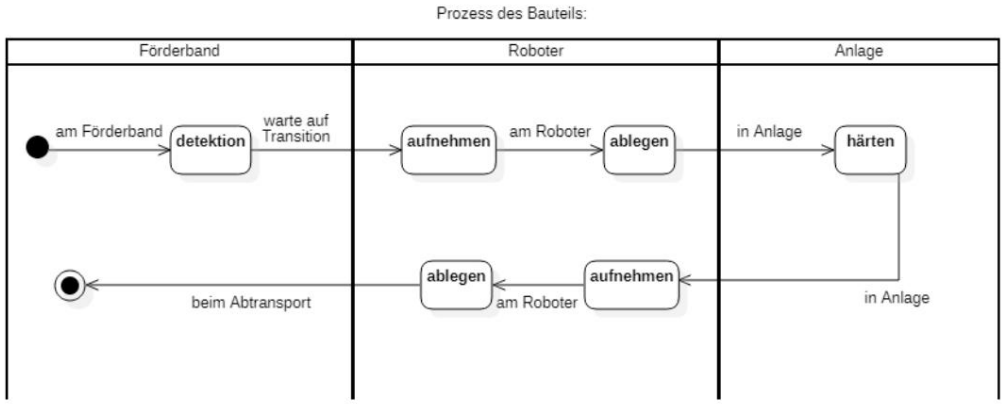



Figure 2.20: Example of an activity diagram in UML

## 2.5 tiling of the plane

Tessellation [9] deals with the arrangement of geometric elements,
to completely cover a plane. It belongs to geometry. The Geometry
is a branch of mathematics and deals with the investigation of variant sizes. Geometric elements are part of this science and will be
described in Euclidean geometry, named after Euclid of Alexandria
(3rd century BC). In mathematics one distinguishes between many types of tiling such as regular and semi-regular tiling, periodic and
aperiodic tiling or platonic tiling. In this work lies the
Focus on the platonic tilings.

### 2.5.1 Platonic tiling

The platonic tiling [10] is of essential importance for the creation of a layout for the charging. It describes 3 possibilities (Fig. 2.21),
completely fill a plane with only one type of regular polygons. With a discretization of the charging levels, these 3 tiling patterns could be used to calculate any size and shape of the to be charged
to fill levels.

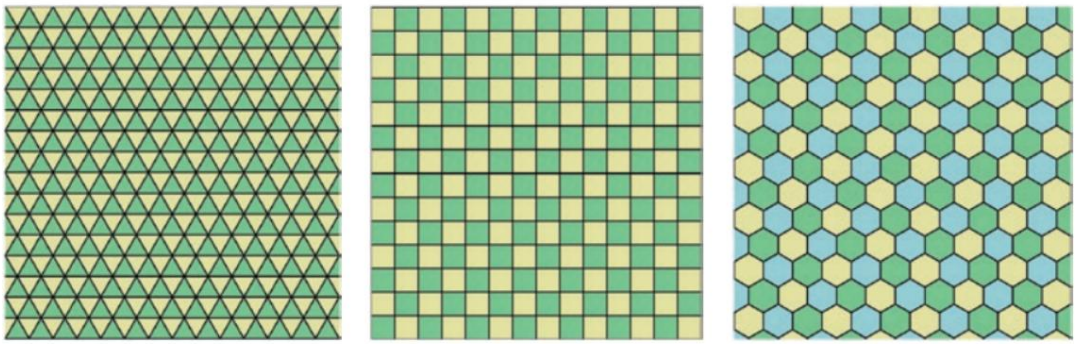


Figure 2.21: Platonic tilings [10]

### 2.5.2 Kepler's conjecture

According to a conjecture by Johannes Kepler, the densest arrangement of spheres in Euclidean space, measured via the mean density, corresponds to a face-centered cubic or hexagonal arrangement [11]. Both types of packing have the same average density and can be considered equivalent.
If one now considers a section through a plane of symmetry of the hexagonal packing, one arrives at Fig. 2.22 and one can assume that this is the closest packing of circles. Now this arrangement could be used for cylindrical components to place the maximum number of this type on a charging rack. The conjecture is now considered proven by [12].

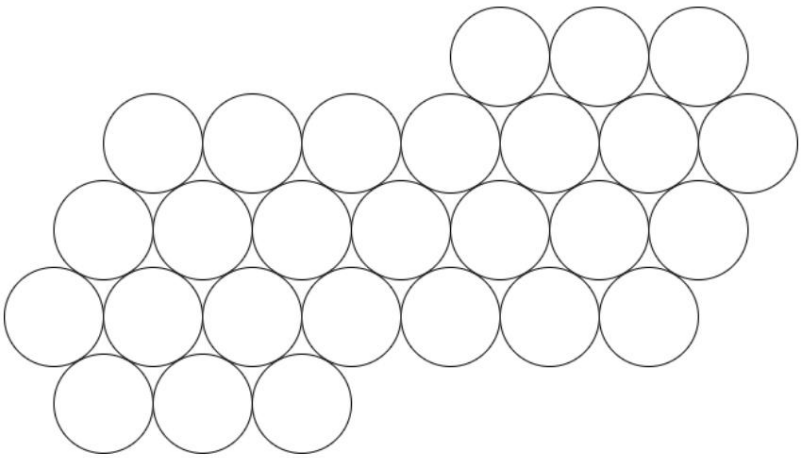


Figure 2.22: Section through a hexagonal packing of spheres

## 2.6 Machine Learning

ML has been seen as a promising technology for some time. In recent years, more and more methods have been developed or further developed to make this technology more versatile. In this chapter, after an introduction to the basics, some of these methods are presented. Basically, ML is divided into 3 sub-areas: Supervised Learning (SL),

Unsupervised Learning (UL) and Reinforcement Learning (RL) (Fig. 2.23). The following explanations were taken from [13], [14] and [15], chapter number and any further references are given in the text. For the planned implementation, the corresponding classes of the library Pytorch [16] presented.

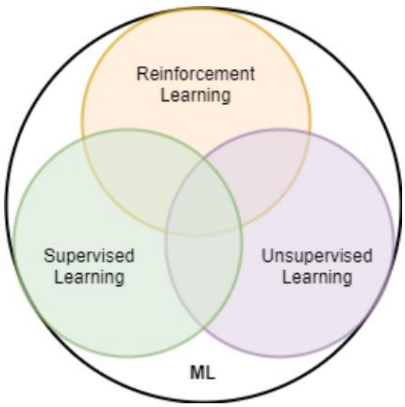



Figure 2.23: Sub-areas of machine learning

### 2.6.1 Basics

In this section, the basics of neural networks are explained, which are necessary to be able to apply the methods of ML. This included the structure of a neuron, activation functions, the gradient method, initialization of weighting factors, cost functionals, regularization, structure a notation and parameter tuning.

#### 2.6.1.1 Structure of a neuron

The neuron [13, chap. 2][15, chap. 10] forms the basic building block of every neural network and is therefore also essential for understanding. a neuron has several inputs, so-called features, which in general start with a

are scalar weighted. These weighted inputs are then summed up and a weighted sum is obtained. Furthermore, to this sum
A constant, real value is added to this, which is assigned to each node individually. We are talking about the so-called bias. To output a value, an activation function, sometimes called a transfer function, is applied to this weighted sum with a direct component. The whole situation is shown in Fig.
2.24 illustrated graphically. Furthermore, it should not go unmentioned at this point remain that the entire transfer behavior, as shown in formula 2.1, can be summarized by matrix multiplication.

$$\hat{y} = f(w^T x + b) = f(z) \tag{2.1}$$

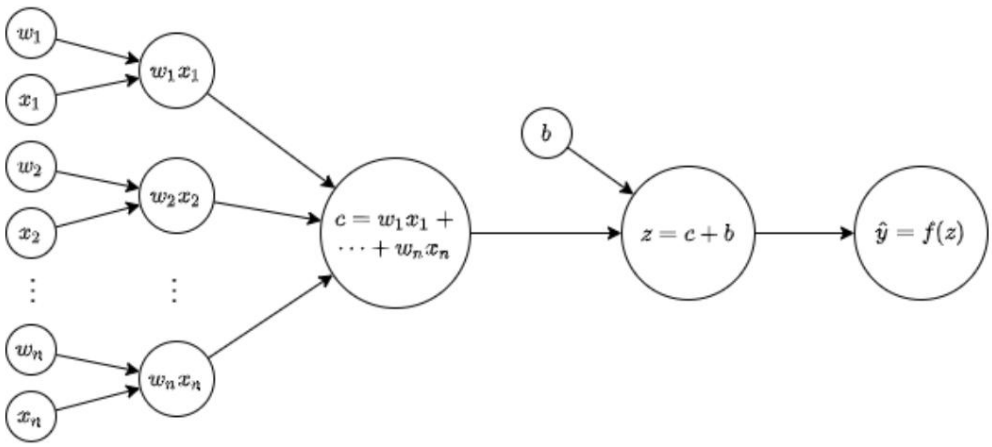



Figure 2.24: Schematic structure of a neuron

A change of the features, i.e. the input variables, is generally referred to as observation. For a clear distinction between features and observations the following convention is now used:

$$x = x_j^{(i)}$$

Here the subscript j stands for the jth feature and the superscript Index i for the it observation. Thus the transfer function of the individual neurons also to:

$$f(e.g.\ x_j^{(i)}) = \hat{y}_j^{(i)}$$

### 2.6.1.2 Activation functions of a neuron

The activation function [13, chap. 2] is used for the value output of a neuron. In the simplest case, this is a unit matrix (Eq. 2.2), with the help of which the weighted sum with the direct component is passed on unchanged. However, there are other activation functions that are more practical have meaning. The 4 most important ones are briefly listed below: Sigmoids (Eq. 2.3), Hyperbolic tangent (Eq. 2.4), Rectified Linear Unit (ReLU) (Eq. 2.5) as well as Leaky ReLU (Eq. 2.6) and Swish (Eq. 2.7). An overview of all activation functions available in Pytorch can be found under [17].

$$f(z) = I(z) = z \tag{2.2}$$

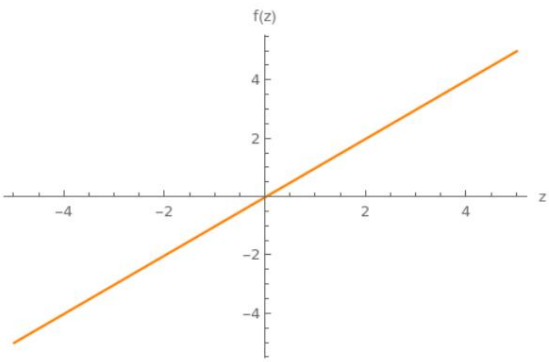



Figure 2.25: Identity activation

$$f(z) = \ddot{y}(z) = 1 \frac{1}{+ \exp(\ddot{y}z)} \tag{2.3}$$

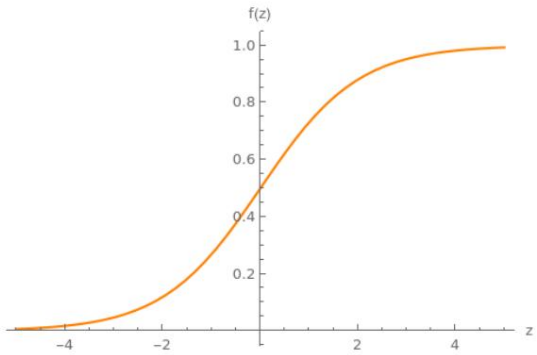



Figure 2.26: Sigmoid activation

$$f(z) = \frac{\exp(z) \ddot{y} \exp(\ddot{y}z)}{\exp(z) + \exp(\ddot{y}z)} \tag{2.4}$$

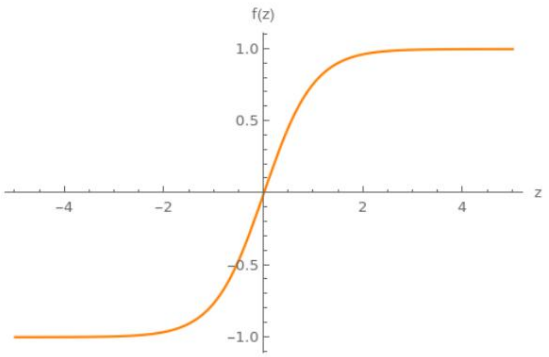



Figure 2.27: TanH activation

$$f(z) = \max(0,z) \tag{2.5}$$

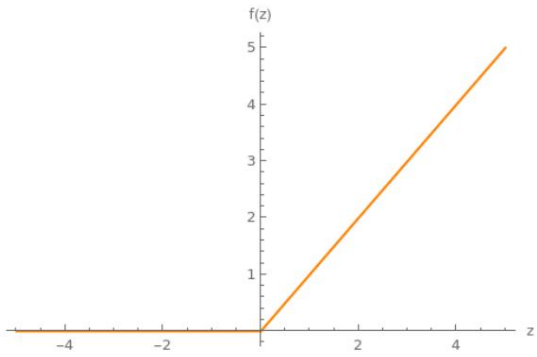



Figure 2.28: ReLU activation

$$f(z) = \begin{cases} \ddot{y}\ \ddot{y}z \text{ if } z < 0 \\ \ddot{y}\ z \text{ if } z \ddot{y}\ 0\ \ddot{y} \end{cases} \tag{2.6}$$

$$f(z) = z\ddot{y}(\ddot{y}z) \tag{2.7}$$

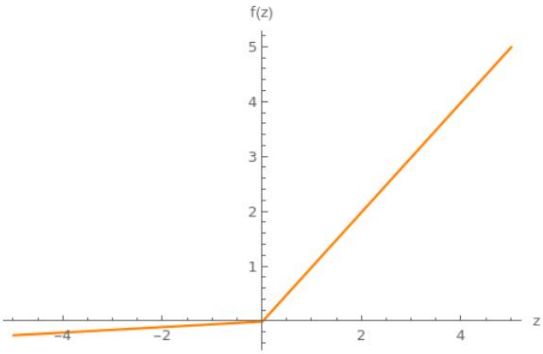



Figure 2.29: LeakyReLU activation

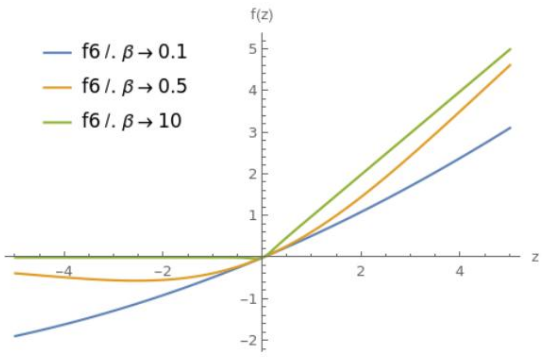



Figure 2.30: Swish activation

#### 2.6.1.3 Determination of the weighting factors

In order to be able to determine the weighting factors for the neurons, a cost functional of the form: J(w) must first be defined. The weighting factors result from a minimization of this cost functional. One also speaks of a minimization in the so-called weight-space. The gradient method [13, Chap. 2] [15, chap. 4] apply (Eq. 2.8).

$$\begin{aligned} \ddot{y}\quad & wn+1 = wn\ \ddot{y}\ \ddot{y}\ddot{y}J(wn) \\ \ddot{y}\quad & bn+1 = bn\ \ddot{y}\ \ddot{y}\ddot{y}J(bn) \\ \ddot{y} & \end{aligned} \tag{2.8}$$

The gradient $\ddot{y}J(w)$ contained in the formula gives this method its name. The gradient, typically denoted as the Nabla operator, gives the direction of steepest ascent of a multidimensional function, where the weighting factors here represent the dimensions. The gradient can be determined by the partial derivatives with respect to the dimension variables (Eq. 2.9).
will.

$$\ddot{y}J(w) = \begin{pmatrix} \ddot{y} & \frac{\ddot{y}J}{\ddot{y}w1}(w) & \ddot{y} \\ & (w)\ \ddot{y}w2 & \\ & \vdots & \\ \ddot{y} & \frac{\ddot{y}J}{\ddot{y}wn}(w) & \ddot{y} \end{pmatrix} \tag{2.9}$$

For the gradient method, the weighting factors are initialized at will at the beginning and then continuously updated through the iteration. This can be carried out until an arbitrary accuracy is reached. In order to prove that the desired accuracy has been reached, a termination condition of the form: |J(wn+1) ÿ J(wn)| < defined. Since the evaluation is usually associated with considerable computing effort, especially in the area of the neural networks, this is often omitted in practical applications and the algorithm is simply executed for a certain number of iterations. The convergence behavior is often graphically represented in a dynamic plot as a cost function over the number of iterations. The parameter ÿ is in the Known in mathematics as the 'relaxation factor', corresponds to the in Newton's method 'Increment', however, is always used in the field of neural networks as learning called rate. In general, this parameter is counted among the hyperparameters, a group of model parameters that are of particular importance for training the networks.

If an iteration consists of a complete pass through the dataset (can consist of many observations), one also speaks of batch gradient descent (BGD) Procedure. If each observation counts as an iteration, one speaks of the stochastic gradient Descent (SGD) method [13, Chap. 3]. As an example: A neural network should Distinguishing between images of cows and goats, for this purpose a data set of 40,000 images may have been prepared. If you now send all 40000 pictures through this network and then updates the weighting factors using the gradient method, this is referred to as a BGD method. If, on the other hand, the weighting factors are updated after each run through of each individual image, one speaks from the SGD process. Now there is still the possibility, in a natural way, introduce an intermediate step, namely dividing the entire batch into so-called minibatches. This possibility is often used in practice and this procedure is called: minibatch gradient descent (MGD) procedure. However, a new hyperparameter is introduced, the batch size. In general, the computing time depends heavily on the calls to the gradient method, so the BGD method is the fastest and the SGD method is the slowest.
The MGD method, depending on the batch size, moves somewhere in between.
However, there are still a number of modified forms of the gradient method [18] [13, chap. 4] [15, chap. 11]. On the one hand there is the momentum method (Eq. 2.10) (Eq. 2.11), which is the past values of the gradient, multiplied by the parameter ÿ, into which the updates are included. The method is less sensitive to saddle points.

$$\begin{aligned} \ddot{y} \quad \ddot{y}w,n+1 &= \ddot{y}\ddot{y}w,n + (1 \ddot{y} \ddot{y})\ddot{y}J(wn) \\ \ddot{y} \quad \ddot{y}b,n+1 &= \ddot{y}\ddot{y}b,n + (1 \ddot{y} \ddot{y})\ddot{y}J(bn) \end{aligned} \tag{2.10}$$

$$\begin{aligned} \ddot{y} \quad wn+1 &= wn \ddot{y} \ddot{y}\ddot{y}w,n \\ \ddot{y} \quad bn+1 &= bn \ddot{y} \ddot{y}\ddot{y}b,n \end{aligned} \tag{2.11}$$

Another method is the RMSProp method (Eq. 2.12 and 2.13).
Compared to the momentum method, two essential extensions were made: Firstly, in the update equation for the method-specific parameter, the gradient was squared and the latter by a

tipplied. On the other hand, in the update equation, the stepping direction was through replaced the gradient divided by a root term. The factor below of the square root remains constant and prevents division by zero. This value usually wears about 10e-8. The basic idea behind this method is that The following: If the ÿ consists of large values, the derivatives are large, so does the parameter S large and thus slows down the learning process, since when S >> 1 also applies $\frac{1}{S}$ << 1 applies.

$$\begin{aligned} \ddot{y} \quad Sw,n+1 &= \ddot{y}Sw,n + (1 \ddot{y} \ddot{y})\ddot{y}J(wn) \ddot{y} \ddot{y}J(wn) \\ \ddot{y} \quad Sb,n+1 &= \ddot{y}Sb,n + (1 \ddot{y} \ddot{y})\ddot{y}J(bn) \ddot{y} \ddot{y}J(bn) \end{aligned} \tag{2.12}$$

$$\begin{aligned} \ddot{y} \quad wn+1 &= wn \ddot{y} \quad \frac{g}{\ddot{y}\sqrt{Sw,n+1+}} \ddot{y}J(wn) \\ \ddot{y}\ddot{y} \quad bn+1 &= bn \ddot{y} \ddot{y} \quad \frac{g}{\sqrt{Sb,n+1+}} \ddot{y}J(bn) \\ \ddot{y}\ddot{y} \end{aligned} \tag{2.13}$$

As a supplement to the optimization methods considered so far, presented another method: Adaptive Momentum Estimation (Adam). This method combines ideas from the optimizers Momentum and RMSProp. Next the method-specific parameters of these optimizers must be determined (Eq. 2.10 and 2.12). The parameters ÿ in the equations are not the same, so they are now denoted by ÿ1 and ÿ2 . Then finds a correction, shown here with the superscript mod, of this parameter instead (Eq. 2.14 and 2.15). Finally, the update for the weighting factors can be calculated (Eq. 2.16). Pytorch allows easy use of these Optimization method (Prog. 2.2).

$$\begin{aligned} \ddot{y} \quad v_{w,n}^{model} &= \frac{v_{w,n}}{1\ddot{y}\ddot{y}\ n_1}) \\ \ddot{y} \quad v_{b,n}^{model} &= \frac{\ddot{y}b,n}{1\ddot{y}\ddot{y}\ n_1}) \\ \ddot{y} \end{aligned} \tag{2.14}$$

$$\begin{aligned} \ddot{y} \quad S_{w,n}^{model} &= \frac{S_{w,n}}{1\ddot{y}\ddot{y}\ n_2}) \\ \ddot{y} \quad S_{b,n}^{model} &= \frac{S_{b,n}}{1\ddot{y}\ddot{y}\ n_2}) \\ \ddot{y} \end{aligned} \tag{2.15}$$

$$\begin{aligned} \ddot{y} \quad wn+1 &= wn\ \ddot{y} \quad \frac{g}{\ddot{y}\ Smod\ w,n+1+}\ v_{w,n}^{model} \\ \ddot{y}\ddot{y} \quad bn+1 &= bn\ \ddot{y}\ \ddot{y} \quad \frac{g}{smod_{b,n}\ +}\ v_{b,n}^{model} \\ \ddot{y}\ddot{y} \end{aligned} \tag{2.16}$$

Program 2.2: Adam class Pytorch

```
1 imported torch
2
3 torches . optimal . Adam ( net . params , lr=0.001 , betas =(0.9 , 0.999) ,
4                           eps=1 e-08 , weight_decay =0 , amdegree = False )
```

#### 2.6.1.4 Initialization of weighting factors

As already mentioned in the previous subchapter, weighting factors can be initialized arbitrarily. However, the random initialization (standard normal distribution N (0, 1)) can lead to numerical problems under certain circumstances
drove [19]. Therefore, suitable initialization strategies [13, Chap. 3] [15, chap. 11] for the respective activation functions or network architectures.
Table 2.3 shows, for example, standard deviations ÿ for the most commonly used normally distributed initializations of the weighting factors, with the expected value µ remaining set to zero. Here nin denotes the number
an input features of the concerned layer and nout the number of output features. For a derivation of the standard deviation see [20], for a compact summary of other initialization [21].

Table 2.3: Initializations for activation functions

| Activation function | standard deviation ÿ for a layer | designation |
|---|---|---|
| sigmoids | $\ddot{y} = q\ 2\ \frac{}{nin+nout}$ | Xavier initialization |
| ReLU | $\ddot{y} = q\ 4\ \frac{}{nin+nout}$ | He initialization |

The Pytorch framework offers an automatic selection of the initialization method
en dependent on the created network. If you want your own specifications, they can
be specified in a separate function in the network class to be created
(program 2.3).

Program 2.3: Weight initialization Pytorch

```
imported torch. nn as nn

class Some_Network ( nn . Module ):
def __init__ (self , input_dim , super ( Some_Network    n_actions ):
        self ). __init__ ()                    ,
        << insert some layers here >>
        self . _create_weights ()


define _create_weights ( self ):
        for m in self . module():
            if isinstance (m              ,    nn . Linear ):
                nn . initial . xavier_uniform_ ( m . weight )
                nn . initial . constant_ ( m. bias              ,    0)
```

#### 2.6.1.5 Cost functional

As a cost functional [13, Chap. 2], those functionals can be used that
that property from Eq. 2.17, i.e. realize a mapping from an arbitrary dimensional space (usually
R n ) to a scalar.

$$J : R^{n} \text{ ÿ } R \text{ with } w \text{ 7ÿ } J(w) \tag{2.17}$$

In the general case, the cost functional does not converge to zero, but
approaches a minimum. The classic square mean value (Mean Square Error (MSE)) is often
used in practice. Written to for the sake of m
observations results from the MSE as in Eq. 2.18 shown.

$$J(wj , b) = \frac{1}{m} \sum_{i=1}^{m} \left( y_j^{(i)} \text{ ÿ } f\, wj , b, x_j^{(i)} \right)^2 \tag{2.18}$$

The term $y_j^{(i)}$ corresponds to the desired target value. The cross entropy (Eq.
2.19) is another cost functional that is used when learning probability statements [22]. As a note
it should be mentioned here that the

The use of this function usually goes hand in hand with the use of the sigmoid activation function, since a value from the interval [0, 1] appears at the output of the neurons. is expected.

$$J(w_j, b) = \frac{1}{m} \sum_{i=1}^{m} L\left(y_j^{(i)}, f\left(w_j, b, x_j^{(i)}\right)\right) \tag{2.19}$$

The so-called Huber loss is another cost functional. For a ¨ **(i)** ÿf wj , b, x (i) more notation, one defines the residual r with r := y − f wj , b, x . Thus the cost functional can be written as follows:

$$J(w_j, b) = \frac{1}{m} \sum_{i=1}^{m} \begin{cases} \frac{1}{2} r^2 & \text{for } |r| \leq \delta \\ \delta |r| - \frac{1}{2}\delta^2 & \text{otherwise} \end{cases} \tag{2.20}$$

The peculiarity here is the quadrature for small residuals and the linear behavior for residuals beyond the threshold value ÿ. However, finding an ideal threshold is not trivial and is the subject of current research [23]. Pytorch again offers a collection of predefined functions that are available in the 'torch.nn' module (Prog. 2.4).

Program 2.4: Cost functional Pytorch

```
imported torch. nn as nn

MSE = nn . MSELoss ( size_average =None 4 CrossEntropy  reduce =None  ,  reduction ='mean ')
= nn . CrossEntropyLoss ( weight =None , size_average =None                                  ,
                                                ignore_index =-100  ,  reduce =None  ,
                                                reduction ='mean ')
HuberLoss = nn . HuberLoss ( reduction ='mean ', delta = 1.0)
```

#### 2.6.1.6 Regularization and Dropout

The regularization [13, Chap. 5][15, chap. 11] is an intervention in the update behavior of the weighting factors. The aim is to counteract the problem of overfitting. By adding the weighting factors to the cost functional, a reduction of the weighting factors to zero when using the optimization methods is prevented. The l1 and l2 regularization are given here as an example ( Eqs . 2.21 and 2.22). Furthermore, a new parameter ÿ is added to describe our model. In Pytorch, this parameter is called 'weight decay' and is transferred when the optimizer is initialized (Prog. 2.2).

Another method to prevent overfitting is the dropout procedure. Here keep probability fur becomes one is defined, which indicates with which probability a neuron knows after completing an iteration step ter remains, i.e. is not deleted. The deletion only takes place during the training, for an application of the trained network to new data, the complete network is used again. Again, Pytorch offers a dedicated class (Prog. 2.5).

$$\tilde{J}(w) = J(w) + \frac{\ddot{y}}{m} \text{kwk1} \tag{2.21}$$

$$\tilde{J}(w) = J(w) + \frac{\ddot{y}}{m} \text{cogeneration}^{2}_{2} \tag{2.22}$$

Program 2.5: Dropout Pytorch

```
1 imported torch. nn as nn
2
3 Dropout = Dropout ( p=0.5 , inplace = False ) 4 test_set =
torch . randn (20 ,20) 5 output = Dropout ( test_set )
```

### 2.6.1.7 Notation for Features and Observations

The entire set of input data from a neuron can be represented as a matrix (Eq. 2.23) [13, Chap. 3], where n describes the number of features and m the number of observations. This notation is important because frameworks like Tensorflow or Pytorch need to create their matrices on initialization.

$$X = \begin{matrix} \ddot{y} \times 1^{(1)} & \times 1^{(2)} & x1\_ & {}^{(m)}\ddot{y} \\ x2\_^{(1)} & x2\_^{(2)} & x2\_ & {}^{(m)} \\ \vdots & \vdots & \ddots & \vdots \\ x\,n^{(1)} & x\,n^{(2)} & x\,n & {}^{(m)} \end{matrix} \tag{2.23}$$

This notation results in a particularly convenient notation for the Argument of the transfer function of individual neurons (Eq. 2.24).

$$z = \text{e.g}^{(1)} \quad \text{e.g}(2) \text{ e.g} \quad (m) = w\,TX + b \tag{2.24}$$

In this way, the transfer functions of the neurons for the entire learning process can be given compactly (Eq. 2.25), where f(z) means the element-wise application to z.

$$\hat{y} = \hat{y}(1)\,\hat{y}(2)\ldots\hat{y}(m) = fz(1)\,fz(2)\ldots fz(m) = f(z) \tag{2.25}$$

In networks with several neurons, which is usually the case in general, it has proved useful to also write the weighting factors as matrices, depending on the current layer l (Eq. 2.26). The dimension of this matrix results from nl ÿ nx: nx corresponds to the number of neurons in the output layer and nl to the number of neurons in the subsequent layer. The bias of all neurons (of the following layer) is also combined in a vector or matrix (Eq. 2.27).

$$W[l] = \begin{pmatrix} w_{1,1}^{(l)} & w_{1,2}^{(l)} & \cdots & w_{1,nx}^{(l)} \\ w_{2,1}^{(l)} & w_{2,2}^{(l)} & \cdots & w_{2,nx}^{(l)} \\ \vdots & \vdots & \ddots & \vdots \\ w_{nl,1}^{(l)} & w_{nl,2}^{(l)} & \cdots & w_{nl,nx}^{(l)} \end{pmatrix} \tag{2.26}$$

$$B[l] = \begin{pmatrix} b_1^{(l)} \\ b_2^{(l)} \\ \vdots \\ b_{nl}^{(l)} \end{pmatrix} \tag{2.27}$$

With this notation one obtains another matrix for the weighted sum z of all neurons of the subsequent layer, starting from the input X (Eq. 2.28). Now this can also be written more generally for any layer (Eq. 2.29), since the input ultimately only represents the preceding layer of the following layer.

$$Z[l] = W[l]X + B[l] \tag{2.28}$$

$$Z[l] = W[l]Z[l-1] + B[l] \tag{2.29}$$

Finally, the transfer functions of the neurons can also be rewritten for all layers (Eq. 2.30), whereby it is assumed here that all neurons use the same activation function and the vector
of the activation functions f only have the same entries.

$$Y[l] = f[l]\, Z[l] \tag{2.30}$$

Since it does not appear to be self-evident, all dimensions of these matrices to remember for the implementation, the dimensions of the matrices for multi-layer networks are summarized again in Table 2.4. The indices l
ranges from 1 to L, the number of layers in the neural network.

Table 2.4: Dimensions of the matrices from this chapter

| matrix | dimension |
|---|---|
| W[l] | nl × nlÿ1 |
| B[l] | nl × 1 |
| Z [lÿ1] | nlÿ1 × 1 |
| Z [l] | nl × 1 |
| Y[l] | nl × 1 |

#### 2.6.1.8 Hyper Parameter Tuning

For the search for a suitable hyperparameters [13, chap. 7] [15, chap. 2], general solution for a black-box problem, in practice one chooses between the following procedures distinguished: grid search, random search, coarse-to-fine optimization and bayesian optimization. The simplest methods are grid-search and random-search. With grid search, the interval from which a parameter is taken
is discretized equidistantly and the system is evaluated for all these parameters.
The extreme value is then assumed to be equal to the largest or smallest of these discrete results. The same procedure is chosen for random search,
with the only difference that the interval for the parameters is not equidistant
is divided, instead values are taken at random from the interval. Interesting
Here is that this method is usually superior to grid search. An improvement on random search is coarse-to-fine optimization. Here the
function, as in our case the cost functional, initially for an arbitrary one

Number of random values evaluated for the parameters. The maximum is then determined from the results. Subsequently, random numbers are again normally distributed with mean equal to extremum generated to function evaluate again. This process is repeated until the point of the extreme value no longer undergoes any significant changes.The spread is your own to draw from experience.

Libraries have also been developed for this, which simplify the implementation of the optimization process. Pytorch recommends the RayTune library for this and explains its application with the help of an example [24]. essentials

Extension of the program code consists in creating a search space as in Prog. 2.6.

Program 2.6: Seach space Raytune

```
from ray import tune
import numpy . random . randint as rndI

search_space = {
    " nodes_layer1 ": tune . sample_from ( lambda _: 2 ** rndI (2 " nodes_layer2 ": tune .      , 9) ) ,
    sample_from ( lambda _: 2 ** rndI (2 7 "lr": tune . loguniform (1 e-4                      , 9) ) ,
                                                    , 1 e-1)    ,
    "
     batch_size ": tune . choice ([2, 4                        , 8th ,  16])
}
```

## 2.6.2 Supervised Learning

In this chapter, the 3 networks: Feedforward Neural Network (FNN), Convolutional Neural Network (CNN) and Recurrent Neural Network (RNN) are presented, which fall within the area SL. The FNN consists primarily of neurons which networked to perform operations on the input data. CNNs leads additional layers, which enable a feature extraction and thus also on raw pixels can be applied. With RNN, the output depends on the previous one state of the features at the entrance, which allows the past to be taken into account.

### 2.6.2.1 Feedforward Neural Networks

In this section, the functionality of FNNs [13, Chap. 3] explained. Under Feedforward is the process of inputting data through a neural network and to send its layers to get values at the output (Fig. 2.31).

The notation required for this was given in Chap. 2.6.1 presented. were thus implicit

also introduced new hyperparameters, namely L the number of layers, ni the Number of neurons per layer and f [l] the activation functions of each Layers.

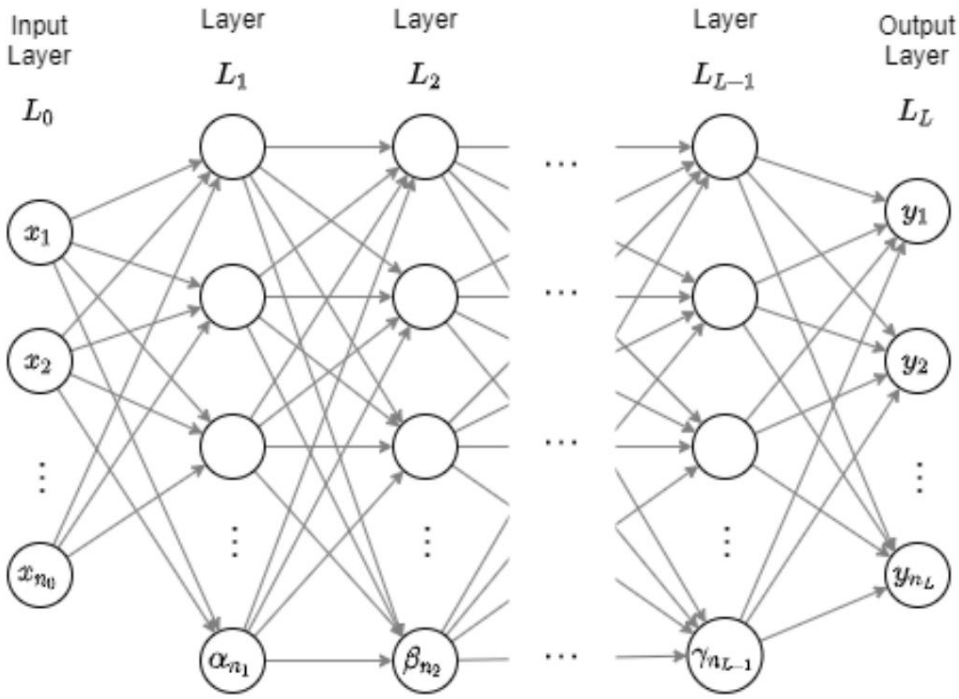



Figure 2.31: FNN

When passing on the weighted summands of the last layer internally, it is important to note that they still have to be normalized for probability outputs, i.e. they have to be projected onto the interval [0, 1]. In addition, the sum of all summands must be 1. These conditions can be fulfilled using a softmax function (Eq. 2.31 and 2.32).

$$S(Z)_i = \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} \tag{2.31}$$

$$\sum_{i=1}^{k} S(Z)_i = \sum_{i=1}^{k} \frac{\exp(z_i)}{\sum_{j=1}^{k} \exp(z_j)} = 1 \tag{2.32}$$

With the knowledge gained so far, one could already implement a classifier using a framework. First of all, however, one should still

Think about the number of fit parameters that can be learned. The number of to
The parameter to be found results for each layer from the number of elements in the
matrix W[l] and the elements in B[l] . The number of degrees of freedom can be
determined from the dimensions given in Table 2.4 (Eq. 2.33). In addition
one can thus calculate the number of degrees of freedom of the entire network
architecture K (Eq. 2.34). No concrete statement can be made about this parameter
about the ute of the network t
kind, but should be considered by the user
be. For example, one could use the information obtained as follows:
Networks are trained with the same number of layers but an unequal number of neurons,
and the favorite is then determined. Then could
one selects the favorite for constant values of K, in networks with the same number
Transform neurons, but of different number of layers, and so on
Carry out optimization steps.
There are separate research papers on the required number of degrees of freedom
Area effective degrees of freedom [25].

$$K[l] = n_l n_{l-1} + n_l = n_l(n_{l-1} + 1) \tag{2.33}$$

$$K = \sum_{l=1}^{L} n_l(n_{l-1} + 1) \tag{2.34}$$

Program 2.7 now shows such a network created in Pytorch. In order to keep the
program code compact, a sequence of linear layers is used for
given class.

Program 2.7: FNN Pytorch

```
imported torch. nn as nn

class FeedForwardNet ( nn . modules ):
def __init__ ( FeedForwardNet , input_dim , output_dim ,
                              nodes_layer1 =120 self ).    , nodes_layer2 =84):
        super ( Net      ,    __init__ ()
        self . ff = nn . Sequential (
                                nn . Linear( input_dim , nodes_layer1 ) nn .                    ,    nn . ReLU ()     ,
                                Linear( nodes_layer1 , nodes_layer2 ) nn . Linear                    ,    nn . ReLU ()     ,
                                ( nodes_layer2 , output_dim ) )

def forward (self                    ,    x):
        return self . f (x )
```

### 2.6.2.2 Convolutional Neural Networks

An extension of the FNN is the CNN [13, Chap. 8] [15, chap. 14]. This
Networks make it possible to efficiently transfer entire frames to a neural network
because there is a feature extraction for larger amounts of data. Around
to be able to go into the difference between the two networks, the
Convolution and pooling are introduced.

#### folding

Convolution means filtering, also known from image processing, using filter matrices
(kernels). The filter matrices are usually square and
odd in their dimension. For the convolution, the matrix to be filtered A vom
Filter kernel K is traversed using a step size s and a new, filtered one
Matrix created (Eq. 2.35). The filter core has the dimension (2na + 1) x
(2nb +1). In neural networks, especially in classification tasks, this
Method used to work out better characteristics from a given data set. Filter kernels that
are frequently used in image processing are: smoothing or averaging filters, sharpening
filters, Laplacian filters, etc. A brief introduction is given here
only the use on two-dimensional matrices, but the method can also be used for
three- or higher-dimensional matrices can be used, with the filter kernel in
is simply adjusted to its dimension. Please note: the number of hyper parameters is
further increased by the introduction of such arithmetic operations into one's own network
architecture.

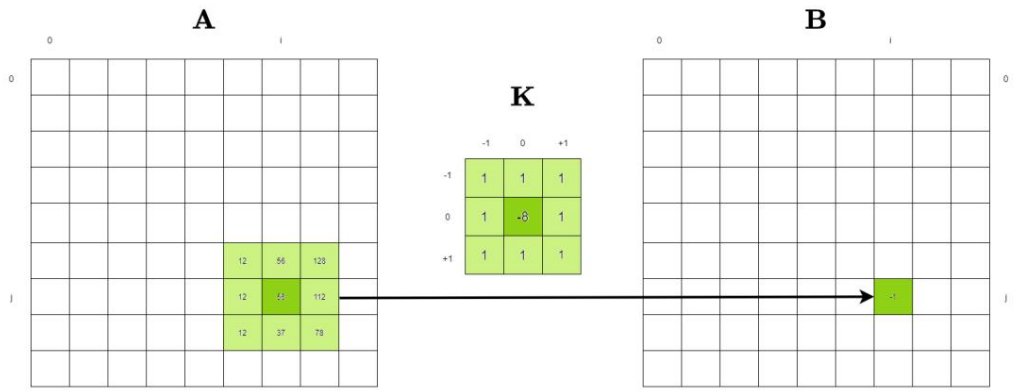



Figure 2.32: Simple Laplace filter kernel

$$B_{i,j} = \sum_{k=-na}^{na} \sum_{l=-nb}^{nb} A_{i+k,j+l} K_{k,l} \tag{2.35}$$

Figure 2.32 shows an application of formula 2.35 with na = nb = 1 and an increment s = 1. The mapping is also a special case of the application (Dim(A) = Dim(B)), since for ¨ i = j = 0 the calculation rule is not used directly can be. Here the so-called padding was applied to the dimension of the input matrix. With padding you expand the input matrix mentally with the required rows and columns. The entries of the column and Although row vectors are arbitrary, they are mostly chosen with zero.
If the step size was not chosen with s = 1, the dimensions of A and B would drift apart. In such a case, the increment of i and j responsible for the dimensionality reduction and the formula 2.35 would have to be rewritten. For further details, however, reference is made to the relevant literature [26].

In order to be able to use these filters, the one shown in program code 2.8 is used class used. The entries of the filter kernel are searched ones weighting factors.

Program 2.8: Convolution layer Pytorch

```
imported torch. nn as nn

conv = nn . Conv2d( in_channels , out_channels , kernel_size , stride =1 ,
                    padding =0, dilation =1, groups =1, bias =True padding_mode ='zeros ,
                    ')
```

**pooling**

Pooling is a similar operation to filtering using Folding. The difference is that in the area of the filter kernel (pool) no arithmetic operation is actually carried out, but the maximum or mean value of the pool is transferred to the output matrix. Here, the resulting dimensionality reduction is desired and is achieved by larger step sizes mostly intensified. A 2x2 matrix is a short example: A is a 4x4 matrix, defined as the pool and s = 2 is chosen as the increment.
Now the pool is run through the input matrix. In Formula 2.36, the The position of the pool is indicated by the numerical values marked in bold. The maximum is determined at each position of the pool and transferred to B (Eq. 2.37). The resulting dimension of B can be determined in advance using Equation 2.38.

$$A = \begin{matrix} ÿ\ 1\ 2\ 3\ 4 & ÿ & ÿ\ 1\ 2\ 3\ 4 & ÿ \\ 5\ 6\ 7\ 8 & & 5\ 6\ 7\ 8 & \\ 9\ 10\ 11\ 12 & & 9\ 10\ 11\ 12 & \\ 13\ 14\ 15\ 16 & & 13\ 14\ 15\ 16 & \end{matrix}$$

$$= \begin{matrix} ÿ\ 1\ 2\ 3\ 4 & ÿ & ÿ\ 1\ 2\ 3\ 4 & ÿ \\ 5\ 6\ 7\ 8 & & 5\ 6\ 7\ 8 & \\ 9\ 10\ 11\ 12 & & 9\ 10\ 11\ 12 & \\ 13\ 14\ 15\ 16 & & 13\ 14\ 15\ 16 & \end{matrix} \tag{2.36}$$

$$A ÿ B = \begin{matrix} ÿ & 6\ 8 & ÿ \\ ÿÿ & 14\ 16 & ÿÿ \end{matrix} \tag{2.37}$$

$$nB = \frac{nA ÿ nK}{S} + 1 \tag{2.38}$$

Finally, the Pytorch class intended for pooling should be mentioned (Prog. 2.9):

Program 2.9: Pooling layer Pytorch

```
imported torch. nn as nn

pool = nn . MaxPool2d ( kernel_size , stride =None , padding =0 ,
                        dilation =1, return_indices =False ceil_mode = False ) ,

```

Now, after this brief introduction, the structure of a CNN can be presented will. Namely, these networks consist of filter layers and an FNN. One Filter layer mostly consists of a convolution, followed by a pooling. Figure 2.33 shows such a network. As already indicated by the designations, several layers consisting of filters and pooling can also be connected in front of the FNN.

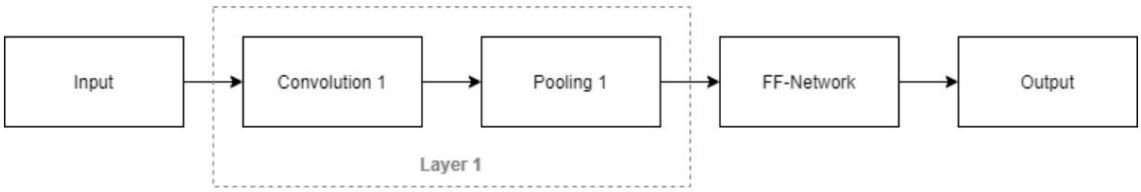



Figure 2.33: Layers of a CNN

Program 2.10 shows the structure of such a network. For this, the program 2.7 was extended by the required layers. The pipeline shown in Fig. 2.33 is mapped by the function 'forward(self, x)'. In addition, Pytorch
the possibility already established architectures like Alexnet, Inception and many to use more [27].

Program 2.10: CNN Pytorch

```
imported torch. nn as nn
imported torches. nn . functional as F

class FeedForwardNet ( nn . modules ):
def __init__ ( FeedForwardNet , input_dim , output_dim ,
                              nodes_layer1 =120 self ).    , nodes_layer2 =84):
        great (Net        ,    __init__ ()
        self . conv1 = nn . Conv2d (3, 6, 5)
        self . pool = nn . MaxPool2d (2 self . conv2    ,    2)
        = nn . Conv2d (6                                ,    16 ,    5)
        self . ff = nn . sequential (
                              nn . Linear( input_dim , nodes_layer1 ) nn .                    ,    nn . ReLU ()     ,
                              Linear( nodes_layer1 , nodes_layer2 ) nn . Linear                    ,    nn . ReLU ()     ,
                              ( nodes_layer2 , output_dim ) )
self . input_dim = input_dim

def forward (self                    ,    x):
        x = self . pool ( F. relu ( self . conv1 (x) )) # Layer 1
        x = self . pool ( F. relu ( self . conv2 (x) )) # Layer 2
        x = x. view (-1, self . input_dim ) # reshape
        return self . f (x )
```

### 2.6.2.3 Recurrent Neural Networks

RNNs [13, chap. 8] differ significantly from the networks discussed so far and are used for data where the order is crucial. Such data could, for example, be sentences like those processed by chatbots. So the input data now consists of sequences and the network

applies its operations to all these sequences. Furthermore, previous sequences are stored indirectly via an internal state in order to form a prediction for the next sequence. Figure 2.34 shows the two
Representation of these networks (1 neuron), on the left the collapsed (folded) and on the right the spread out (unfolded). x describes the input data or
Sequences, for example the letters of a word. Each sequence is in its
The order is transferred to the network, thereby changing the status of the current one layer, but not their weighting factors. Will now be another sequence
transferred to the network, the output is significantly influenced by the current state. The order of the sequences is identified by the index t. how
As can be seen in the figure, two sets of weighting factors play a role here
Rolle, W and U

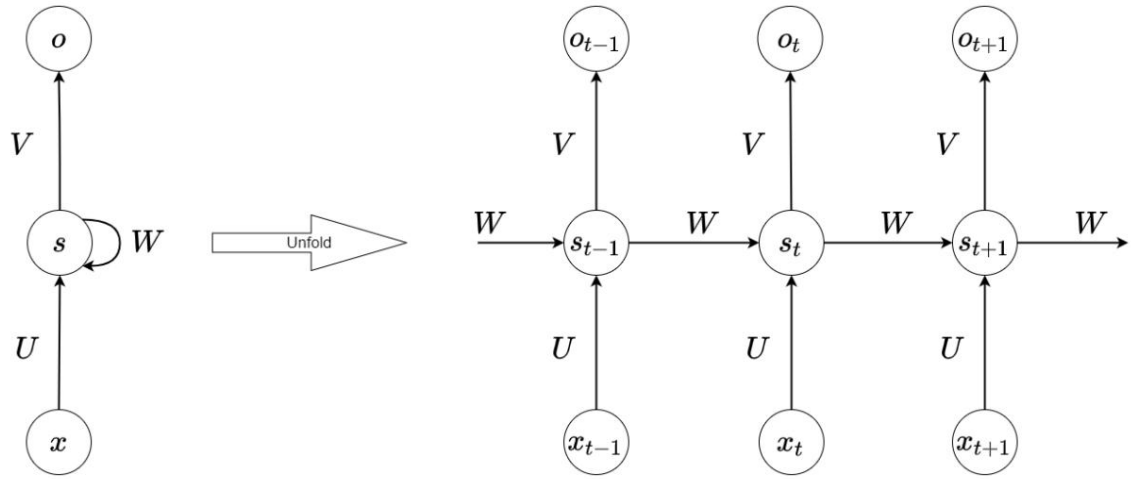



Figure 2.34: Recurrent network

The internal state is calculated using these two weighting matrices.
Mathematically, one could reproduce the relationship as shown in Formula 2.39. Here you can also see the reason for the designation recurrent, because the
State is defined recursively, so it depends on all previous values.

$$st = f(Uxt + Wstÿ1) \tag{2.39}$$

Areas of application are: text generation, translations, speech recognition, chatbots and much more. There are also more sophisticated neurons for this type of network, such as Long-Short-Term Memory (LSTM) and Gated Recurrent Unit (GDU), which can also be found in source codes from
RL agents can be found again [28].

### 2.6.3 Unsupervised Learning

The UL area [15, Chap. 9] deals with compression and segmentation tasks (clustering). Images are mostly used as input, as is also the case
is the case with CNNs. A possibility of implementing the segmentation task in Pytorch is presented by [29]. The author implemented a ¨
encoder and decoder class, which are merged into an autoencoder class ¨
became. This class was subsequently used in a training routine to
adjust the parameters of the encoder class. The encoding depends on
the VGG-16 architecture (CNN). Clustering was done in the next step using
implemented using the Local Aggregation (LA) method, which defines a function
which indicates how well a collection of extracted features fits into a group.
The implementation of this method is consistently challenging, so here
no further considerations are given, as the UL range is only mentioned here for
completeness.

### 2.6.4 Reinforcement Learning

In this chapter, the basis of RL [14, Chap. 1] in order to be able to apply methods from this area in concept development. As an introduction ¨
An introduction to the most important terms follows, and these terms are then placed in context with each other. Subsequently, the details in the
respective subchapters

- Agent: A construct to interact with the environment.
- Environment: The environment in which an agent can perform an action.
- Action: A possible action that the agent can perform in the environment.
- Current state: Information that is transmitted to the agent.
- Policy: The rule that the agent learns. It describes what action at
  to be executed in a state.
- Reward: The reward or punishment resulting from an action.
- New state: The state that the environment assumes after the action has been
  taken. An observable part of the state corresponds to an observation.
- Value: The value of a state. It consists of the sum of all the following
  Rewards and reflects the severity of the state when the interaction proceeds
  according to the chosen rule.

In summary: Value comes from rewards, which you receive from actions. The agent collects these rewards while following a certain policy and sums them up to determine the value of a current state. After the current state value has been determined, the agent changes its policy so that it maximizes this value.

#### 2.6.4.1 Policy

A distinction is made between deterministic (Eq. 2.42) and stochastic policies (Eq. 2.41) [14, chap. 2]. By deterministic is meant that each state can be assigned an optimal action that depends only on the current state. The term stochastic is used to express that actions may initially result in a loss, but in the long run increase the overall gain. In favour of which action the decision is made is printed out by the policy (Eq. 2.40). The facts can, as shown in Fig. 2.35, graphically

be illustrated.

$$\ddot{y} = p(a|s) \tag{2.40}$$

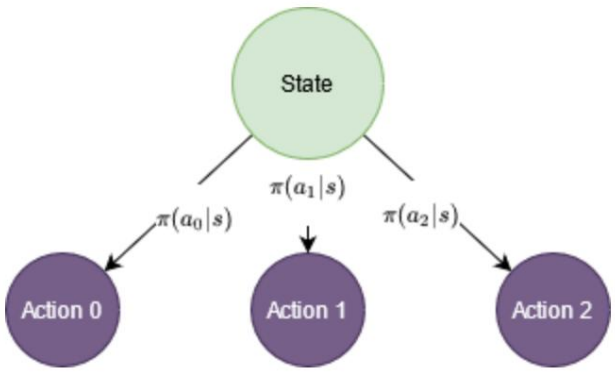



Figure 2.35: Choosing an action using policy

$$\{(\ddot{y}(a0|s), \ddot{y}(a1|s), \ldots, \ddot{y}(an|s)) : \ddot{y}(ai|s) \ddot{y} [0, 1[, \sum_{i=0}^{n} \ddot{y}(ai|s) = 1\} \tag{2.41}$$

$$\{(\ddot{y}(a0|s), \ddot{y}(a1|s), \ldots, \ddot{y}(an|s)) : \ddot{y}(ai|s) \ddot{y} \{0, 1\}, \sum_{i=0}^{n} \ddot{y}(ai|s) = 1\} \tag{2.42}$$

The probability of a condition and reward occurring when the
the current state and the selected action are known, is defined by the transition
described dynamically (Eq. 2.43) [14, Chap. 2]. Here St and Stÿ 1 correspond to the set
of all possible states, Atÿ1 corresponds to the set of all possible actions.
Spoken the formula means: The probability that in the next step
the environment the condition s ° accepts and results in a reward r,
if the last state was s and action a was performed. The formula lets
can also be represented graphically as shown in Fig. 2.36, one speaks of one
so-called backup diagram.

$$p(p^{\circ}, r) = P(St = s^{\circ}, Rt = r|Stÿ1 = s, Atÿ1 = a) \tag{2.43}$$

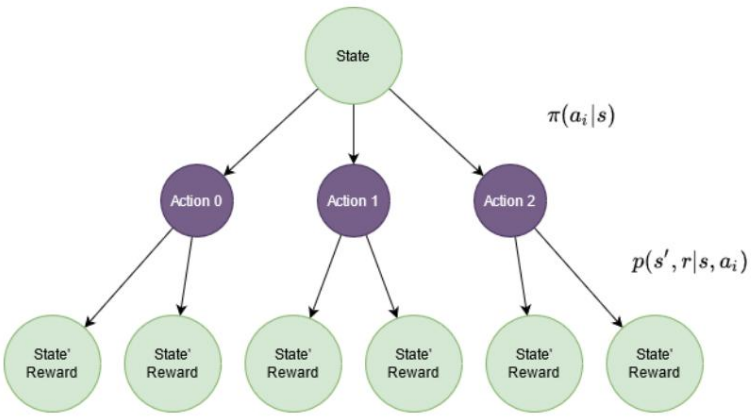



Figure 2.36: Backup diagram

### 2.6.4.2 Rewards and Exploration Versus Exploitation Dilemma

To control the effect of the reward on AI training, use something similar to
introduced a discount factor for the calculation of interest [14, chap. 2]. The discount factor
indicates how the value of a reward changes over time. So
like in the financial world, an amount of money is immediately worth more than the same amount of money
later, so when learning, a quick reward is often preferable to a later one. Therefore, later
rewards are weighted differently than immediate ones. at
Furthermore, for continuous tasks, the sum of the rewards could go to infinity, leading to
numerical problems. If you now choose the discount
factor ÿ = 0, so the AI becomes very short-sighted about rewards. One chooses
On the other hand, if ÿ = 1, a certain far-sightedness will appear during training.

$$Gt = Rt+1 + ÿRt+2 + ÿ\,2Rt+3 + \ldots + ÿ^{T}\,ÿ1RT \tag{2.44}$$

With a given policy, the agent runs through certain states whose values it tries to determine. Based on the determined values, he changes the policy to a better one. However, there can be states that the agent never observes could, but exist in the area, not be taken into account. This is referred to as the so-called exploration versus exploitation dilemma [14, Chap. 2, 9]. One One way to get around the limited field of view effect is to perform a random action that doesn't follow the policy. How should he How does the decision-making process for these actions look like? Among other things, the epsilon greedy policy was developed for this purpose (Eq. 2.45).

$$\ddot{y}(a|s) = \begin{cases} a = \operatorname{argmax}_a Q(s,a) & 1 \ddot{y} \\ a = \text{edge}_a & \ddot{y} \end{cases} \tag{2.45}$$

With the epsilon greedy policy, the agent performs the action with the highest q value with a probability of 1ÿ. A random action is played with the probability of being performed and is shared equally among all actions to ensure that the agent explores non-maximizing actions. This Facts are generally known under the term off-policy. The parameter is another hyperparameter and can be changed during training. Other policies such as upper confidence bound or Thompson sampling can be used.

#### 2.6.4.3 Markov Process

The Markov process [14, chap. 2] is divided into 3 areas: Markov chain, Mar kov reward process and Markov decision process. All of these processes involve the Coincidence as a common constant. This coincidence, described by probability values, remains unchanged and the processes are thus independent of the Past. In the following paragraphs, the 3 processes are explained in more detail using a small example considered.

**Markov chain**

The Markov chain describes the probability of a transition to another state. A short example is considered for this purpose: Fig. 2.37 shows four states and the transition probabilities p with which one can enter the respective changes states. The four states are: s0 corresponds to the initial state, s1 and s2 correspond to further states that can be reached from the initial state and s3 represents the final state.

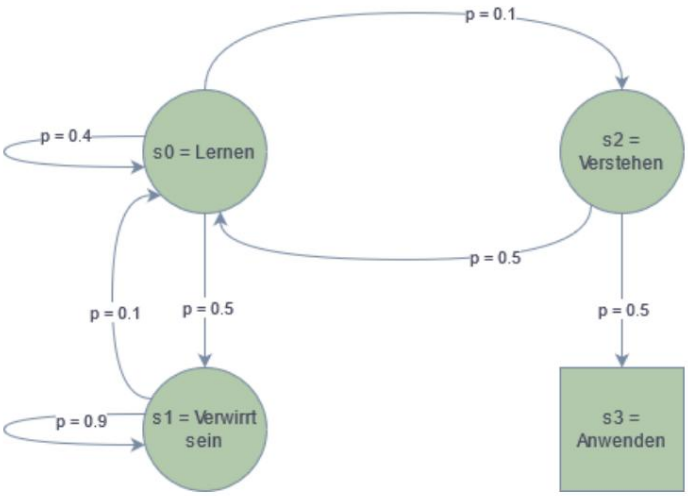



Figure 2.37: Example of a Markov chain

This situation can also be described mathematically (Eq. 2.46). the The probability of a transition depends only on the previous state. Thus, the probability can be expressed as: The probability that the state s ₀ is assumed when the current state is equal to s. These probabilities can be summarized in a transition probability matrix (Eq. 2.47).

$$pi = P(St+1 = s_0 | St = s) \tag{2.46}$$

$$P = \begin{pmatrix} 0.4 & 0.5 & 0.1 & 0 \\ 0.1 & 0.9 & 0 & 0 \\ 0.5 & 0 & 0 & 0.5 \\ 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.47}$$

This example is a periodic Markov chain with a final state. Since the final state, starting in the initial state, must always occur after a finite series of transitions, this is generally referred to as an episode. Different episodes might look like this:

Episode 1 : s0 ÿ s0 ÿ s1 ÿ s1 ÿ s1 ÿ s1 ÿ s0 ÿ s2 ÿ s3

Episode 2 : s0 ÿ s1 ÿ s0 ÿ s1 ÿ s0 ÿ s2 ÿ s0 ÿ s1 ÿ s0 ÿ s2 ÿ s3

### Markov Reward Process

The Markov chain example is used to illustrate the Markov reward process. Now, in this example, rewards for a Transition introduced (Fig. 2.38). A value for the individual states can thus be determined (Eq. 2.48).

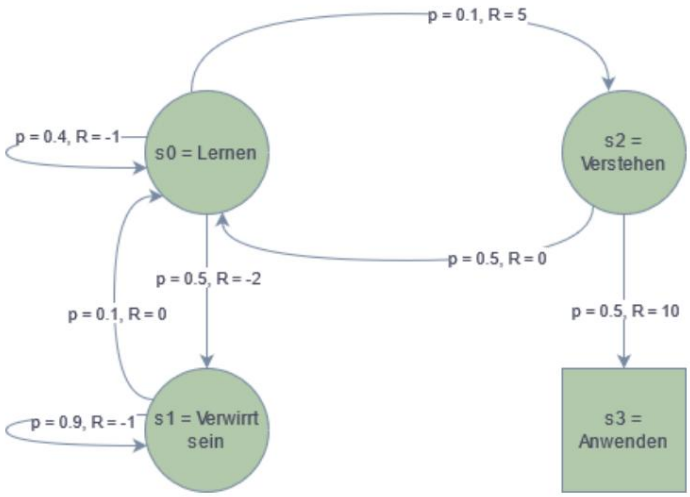



Figure 2.38: Example Markov reward

$$v(s) = E[Gt \mid St = s] \tag{2.48}$$

The value of a state results from the expected value E of Gt (Eq. 2.44) if the state s prevails. However, the expected value is in practice replaced by an average over many episodes (Monte Carlo simulation). the Transition probabilities and the rewards of the example in Eq. 2.48 inserted gives:

$$v(s0) = 0.1 \ddot{y} 5 + 0.5 \ddot{y} (\ddot{y}2) + 0.4 \ddot{y} (\ddot{y}1) = \ddot{y}0.9$$

$$v(s1) = 0.9 \ddot{y} (\ddot{y}1) + 0.1 \ddot{y} 0 = \ddot{y}0.9$$

$$v(s2) = 0.5 \ddot{y} 0 + 0.4 \ddot{y} 10 = 5$$

### Markov Decision Process

The Markov decision process will now be considered using the example from Markov chain and Markov reward process. Since states are not always at the whim of the change nature, but usually be influenced by actions

one can add these possible actions to each state. For example, one can define the following actions for the state s0 : a0 do nothing, a1 read a book and a2 testing (Fig. 2.39). If a decision is made for the action a0 , then this results neither in a reward nor in a punishment, but it draws the Episode only in length. This has a negative impact on Gt . If the action a2 is chosen, the chance of transition to state s2 is 30% and one becomes also rewarded. The action a1 could also be chosen. With that you would have even a 60% chance of reaching state s2 and getting closer to the goal move. However, there is also a 20% probability that you will switch to state s1 and collect minus points there. So what actions are to be taken now? Thus one ends up again with the description of a policy as it is in Formula 2.43 was given. This now results in a value for the respective state, which depends on the policy used (Eq. 2.49).

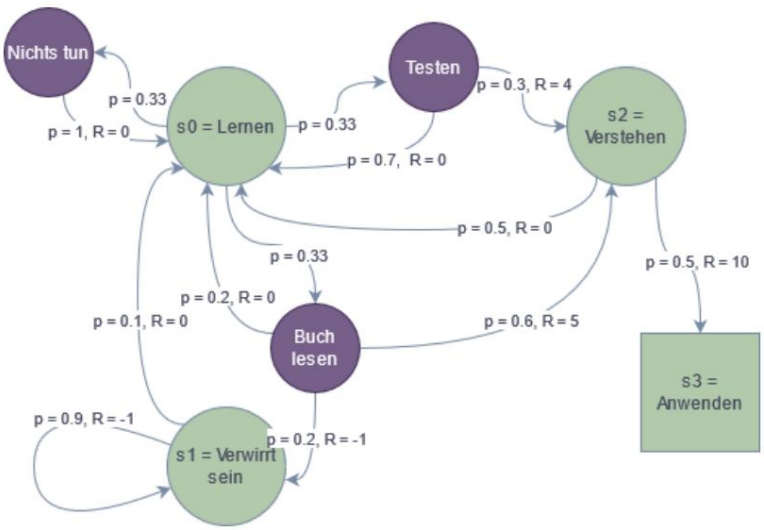



Figure 2.39: Example Markov process

$$v\ddot{y}(s) = E\ddot{y}[Gt \mid St = s] \tag{2.49}$$

The value of a state now results from the expected value Eÿ of Gt (Eq. 2.44) and depends on the chosen policy. Another concept, besides the value function is the action value function (Eq. 2.50). Imagine that it is up to the agent at time t which action it carries out and considers that the policy must then be observed for further times.

$$q\ddot{y}(s, a) = E\ddot{y}[Gt \mid St = s, At = a] \tag{2.50}$$

#### 2.6.4.4 Bellman equation

In this chapter, using the value function (Eq. 2.49) and the discounting function (Eq. 2.44) derived the Bellman equation [14, Chap. 2], which plays an important role in the RL area. To start, the equations are here messaged again.

$$v\ddot{y}(s) = E\ddot{y}[Gt|St = s]$$

$$Gt = Rt+1 + \ddot{y}Rt+2 + \ddot{y}\,2Rt+3 + \ldots + \ddot{y}^{T}\,\ddot{y}1RT$$

The discounting function can be rewritten as:

$$Gt = Rt+1 + \ddot{y}Rt+2 + \ddot{y}Rt+3 + \ldots + \ddot{y}^{T}\,\ddot{y}2RT$$

If you still t $^{\circ} = t + 1$ as a new abbreviation, one obtainsfor the Klam- expression:

$$Gt = Rt\,0+1 + \ddot{y}Rt\,0+2 + \ddot{y}\,2Rt\;\;0+3 + \ldots + \ddot{y}\;\;T\;0\ddot{y}1RT0$$

Substituting this fact into the original equation, one obtains further:

$$Gt = Rt+1 + \ddot{y}Gt+1$$

This fact can in turn be used in the value function:

$$v\ddot{y}(s) = E\ddot{y}[Rt+1 + \ddot{y}Gt+1|St = s]$$

The expected value Eÿ is formed from the sum of all possible actions a, which the agent can assert in state and about all further states, in which the environment transitions to the agent, which are defined by the transition function p(s 0 , r|s, a) . Substituting this definition for the expectation value (Eq. 12.1) leads to the expanded form of vÿ(s), the Bellman equation.

$$v\ddot{y}(s) = \sum_{a} \ddot{y}(a|s) \sum_{s^{\circ},r} p(p^{\circ}, r|s, a)(r + \ddot{y}v\ddot{y}(s^{\circ})) \tag{2.51}$$

The equation can now be interpreted as follows: The value of a Zu
state s is the mean (a weighted sum) over all rewards and state values of the following
states s cy when performing an action. The strategy followed by the transition probability
p(s 0 , r|s, a) based on the state action pair (s, a) (Figure 2.40). The Bellman equation of the
current state with the existing value of the next state (Equation 2.50). A similar relationship

$$q_\pi(s, a) = E_\pi[G_t | S_t = s, A_t = a]$$
$$= E_\pi[R_t + \gamma G_{t+1} | S_t = s, A_t = a]$$

you get:

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)(r + \gamma v_\pi(s'))$$

A relation between vÿ(s) and qÿ(s, a) can also be given since
both are linked by the policy ÿ(s|a). This results in f ¨ vÿ(s): ur

$$v_\pi(s) = \sum_a \pi(a|s) q_\pi(s, a) \tag{2.52}$$

This fact can now be used again in the equation of qÿ(s, a).
and we arrive at the Bellman equation for ¨ qÿ(s, a).

$$q_\pi(s, a) = \sum_{s', r} p(s', r|s, a)\, r + \gamma \sum_{a'} \pi(a'|s') q_\pi(s', a') \tag{2.53}$$

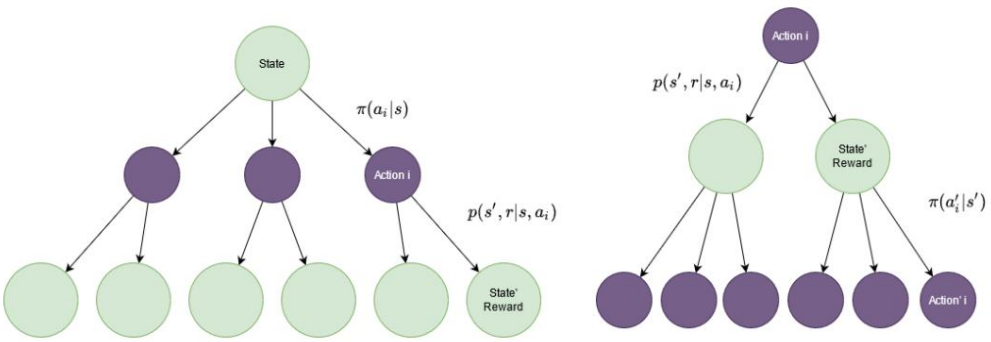



Figure 2.40: Backup diagram for ¨ state and action values

#### 2.6.4.5 Solving the Bellman equation

Solving an RL problem means finding a policy with which the state value function is maximized. If the agent follows the optimal policy, then in state s it will choose that action which also maximizes the action value function q(s, a).
In the following example (derived from [14, chap. 2]) these connections are shown described mathematically:

$$v(s)\ddot{y} = \max_{\ddot{y}} v\ddot{y}(s)$$

$$v(s)\ddot{y} = \max_{a} q\ddot{y}(s, a)$$

$$v(s)\ddot{y} = \max_{a} \sum_{s',r} p(p^{0}, r|s, a)\, r + \ddot{y}v(s^{0})\ddot{y}$$

Now the example from Markov process is continued at this point. One can all combinations of: current state, action, reward and new to stood as summarized in Table 2.5.

Table 2.5: All possible (s,a,s',r) combinations from Fig. 2.39

| State(s) | Action (a) New State (s') Reward (r) Transition p(s 0 , r\|s, a) | | | |
|---|---|---|---|---|
| s0 learning | a0 doing nothing s0 learning | | 0 | 1.0 |
| s0 learning | a1 testing | s0 learning | 0 | 0.7 |
| s0 learning | a1 testing | s2 understanding | 4 | 0.3 |
| s0 learning | a2 read a book s0 learn | | 0 | 0.2 |
| s0 learning | a2 read a book s1 be confused -1 | | | 0.2 |
| s0 learning | a2 reading a book s2 understanding | | 5 | 0.6 |
| s2 understanding | - | s0 learning | 0 | 0.5 |
| s2 understanding | - | s3 Apply 10 | | 0.5 |
| s1 be confused - | | s2 learning | 0 | 0.1 |
| s1 be confused - | | s1 Be confused -1 | | 0.9 |

With the help of this table, the state values of the states can now be determined according to the above calculation rule. Here the discount factor is still undetermined.

$$v(s0) = \max \begin{cases} a0: 1\,0 + \gamma v(s0) \\ a1: 0.7\,0 + \gamma v(s0) + 0.3\,4 + \gamma v(s2) \\ a2: 0.2\,0 + \gamma v(s0) + 0.2\,1 + \gamma v(s1) + 0.6\,5 + \gamma v(s2) \end{cases}$$

$$v(s1) = 0.1\,0 + \gamma v(s0) + 0.9\,1 + \gamma v(s1)$$

$$v(s2) = 0.5\,0 + \gamma v(s0) + 0.5\,10 + \gamma v(s3)$$

In order to obtain a simple solution to the problem, $\gamma$ can be set to zero. Thus, a short-sighted behavior of the agent would be expected, since only the prospect of a quick reward is taken into account. The equations are therefore no longer recursive and have a constant value from the start.

Table 2.6: Solving MDP with $\gamma = 0$

| $v_\gamma$ | t | t+x | t+y |
|---|---|---|---|
| v(s0) | a1 0.12 | a1 0.12 | a1 0.12 |
| v(s1) | -0.9 | -0.9 | -0.9 |
| v(s2) | 5 | 5 | 5 |

From this calculation it can be concluded that the action 'test' should always be chosen if only quick rewards are desired. At this point it should be noted that action a1 is just about the best choice, action a2 would result in a state value of 0.1. If one were to try to solve the system $\gamma = 0$, one would find that the recursive (Fig. 2.4) and original equations lead to problems. Therefore for

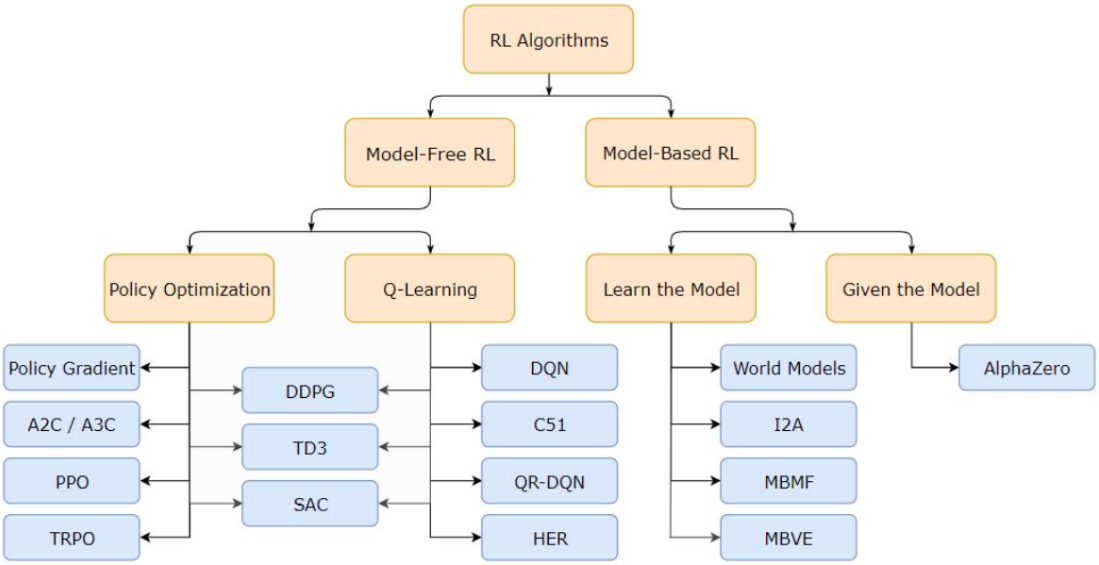



Figure 2.41: Algorithms for RL [30]

#### 2.6.4.6 Temporal Difference Learning

Temporal Difference Learning (TDL) [14, Chap. 4] combines the advantages of dynamic programming and Monte Carlo simulation. For this, bootstrapping from dynamic programming and sampling from the Monte Carlo method are used.
equation 2.54 shows the scheme for updating the state-values.

$$V(s) \leftarrow V(s) + \alpha \left[ R + \gamma V(s') - V(s) \right] \tag{2.54}$$

The instantaneous estimated value for the cumulative rewards resulting from state s is now bootstrapped from the estimated value for the subsequent state $s'$, obtained through several runs (sample-based). If one replaced the term $R+\gamma V(s')$ with $G_t$ (Eq. 2.44), one would have followed a pure Monte Carlo approach. The update shifts the estimated value from V (s) to $V(s) + \alpha\delta_t$ where $\delta_t$ corresponds to the temporal difference error (Eq. 2.55).

$$\delta_t = R_{t+1} + \gamma V(s_{t+1}) - V(s_t) \tag{2.55}$$

#### 2.6.4.7 Deep Q Learning

Deep-Q Learning (DQL) [14, Chap. 6] is a model-free, off-policy, temporal-difference learning method in the field of RL. This method involves an agent who follows a certain policy and gains experience. These consist of a tuple of the form: (current state, action taken, resulted reward, next state, end of game? Y/N ). The agent then uses this experience and the Bellman equation to determine, in an iterative loop, the optimal policy that maximizes the value function of each state. The optimization is done by the update rule given by Eq. 2.56. The extraction of experiences, stored in the so-called replay buffer, can be done in different ways. Usually, for the sake of simplicity, a normal distribution is chosen for the selection, but these could also be prioritized [31].

$$Q(s, a) \mathrel{!}= Q(s, a) + \ddot{y}\, h\, R + \ddot{y} \max_{0a} Q(p^{0}, a0) \ddot{y}\, Q(s, a)\, i \tag{2.56}$$

In order to be able to use this method for environments with a large number of states, the q-value of a policy is estimated by a function approximation (deep learning) $\tilde{q}(s, a, w) \ddot{y} q\ddot{y}(s , a)$. Thus the q-value depends on the weighting factors and one speaks of the DQL. Taking linear approximation and the BGD method from Chapter 2.6.1 into account, an update rule for the weighting factors can now be determined (Eq. 2.57).

$$1\ wt+1 = wt + \ddot{y}\, \frac{}{n} \sum_{i=1}^{n} hri + \ddot{y} \max_{hey} q\tilde{}(si^{0}, a0, wtarget) \ddot{y} q\tilde{}(si , {}^{hey}, wononline_{t}) i\ \ddot{y} wq\tilde{}(si , {}^{hey}, online_{t}) \tag{2.57}$$

Index t points to the optimization step and index i to the Po target remain in the batch for numerical reasons. The weighting factors w constant in intervals and is marked with w at the end of the interval $w_{t}^{online}$ overwritten.

#### Double Deep Q Learning

Equation 2.57 shows that the same network is used to determine the maximizing action and to determine the q-value. Written in simplified form, this corresponds to:

$$Y\ DQN\ q\tilde{}(s = r + \ddot{y} \max_{0a} {}^{0}, a0, w\, t^{target})$$

The equation can also be given as equivalent to:

$$Y^{DQN} = r + \gamma \tilde{q}(s', \operatorname{argmax}_{a'} \tilde{q}(s', a', w_t^{target}), f_t^{target})$$

Now we first take the maximizing action and then determine the q value. This is equivalent to the previously chosen procedure, directly den to determine the maximum q-value. Now you can see that the same weighting factors are used to determine the best action and to calculate the q-value be used. This leads to the so-called maximization bias, which [32] was investigated. The authors of the paper recommend an approach that they call double DQN, where the weights for determining the best argmax $a' \tilde{q}(s', a')$ with $w^{on-line}$ action are chosen and the weights for the determination of the q-value with $w^{target}$.

$$Y^{DQN} = r + \gamma \tilde{q}(s', \operatorname{argmax}_{a'} \tilde{q}(s', a', w_t^{nonline}), f_t^{target})$$

**Dueling Double Deep Q Learning**

Until now, the networks always took a state as input and gave q-values Q(s, a) for all possible actions a. Often, however, some actions in a particular state do not have a significant impact. Imagine yourself Car ahead on an empty lane, as long as no obstacles appear, is one small change in direction or speed meaningless. Thus, all of these actions result in similar $^{primal}$ q-values. Now the question is, is it possible to determine the average value of a state and the advantage of a particular one Separate action beyond this average? This procedure was examined by [33]. The authors showed that the behavior of the network such an approach can be significantly improved, especially when the number of possible actions increases. A new parameter A was introduced for this purpose, the so-called advantage (Eq. 2.58).

$$A^{\gamma}(s, a) = Q^{\gamma}(s, a) - V^{\gamma}(s) \tag{2.58}$$

$V^{\gamma}(s)$ measures the value of being in a certain state, $Q^{\gamma}(s,a)$ gives the value to choose a specific action in this state. The difference of the two Value can now be interpreted as an advantage. The authors of the paper developed a network that splits the status at the input into two data flows.

One to determine the state-values V(s) and the other to determine the advantages A(s, a). Finally, both data flows are brought together again to get the q-values. For an improved stability of the procedure the authors subtracted the mean value of advantage from the end nodes Q(s, a) (Eq. 2.59). An illustration of the data flow for this network is shown in Fig. 2.42. The 'Preprocessing' block hides operations that be implemented by convolution layers or pooling layers. As 'input data' could e.g. B. raw pixels serve.

$$Q\hat{}(s, a, w1, w2, w3) = V\hat{} (s, w1, w2) + A\hat{}(s, a, w1, w3) \ddot{y} \text{ mean}(A\hat{}) \quad (2.59)$$

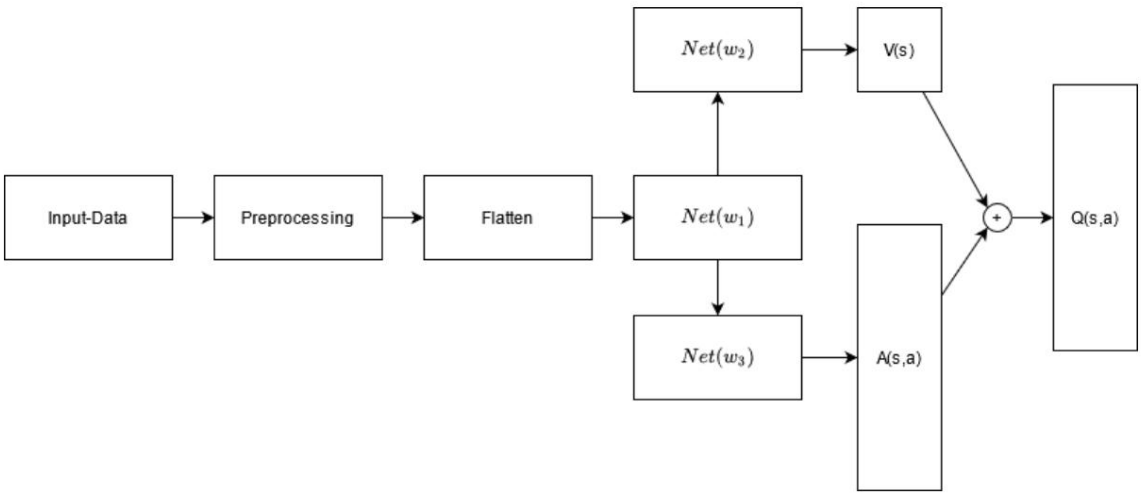



Figure 2.42: Dueling Double Deep Q Network (DQN)

# 3 concept development

For the realization of the objectives a minimalistic tool for the system configuration is developed in Python. In particular, the Tkinter library, which is included as standard in Python, is used for this purpose. With the help of this library, an interaction with the user is to take place in order to
to pass the necessary configuration parameters. Furthermore, the software should contain a simulation to illustrate the configured process. In
This simulation should also show how the created neural network works
be apparent as the environment is specifically designed for this reason.
At the beginning, the workflow is presented from the point of view of the component, from which classes can be derived that enable a corresponding realization of the process. In the context of reliability and flexibility, we will point out
the corresponding attributes, methods and states for the
respective classes defined. Furthermore, a superclass is developed that contains attributes and methods that are required by all classes.
Section 3.4 describes the procedure for developing the tiling process. For this purpose, the area of application is limited to cylindrical components and components in the form of tetrominoes [34]. Tetrominos are geometries that can be joined together from four squares, as some might find in the classic game
Tetris are known.
In the last section, the concept for the control interaction is presented.
The created software should be able to communicate with a PLC from Siemens (S7), the OPC UA communication protocol presented in Chapter 2.3 is used for this. A Kuka training cell is available to validate the concept
Disposal. Since the system training cell primarily consists of only one turntable and one robot, a simplified version of the software is produced for validation. It is assumed that the results will subsequently relate to the
transfer the full version.

## 3.1 Workflow

In order to get an overview of the components and their attributes and methods, a component is mentally sent through the system process, as is also the case in the activity diagram (Fig. 2.20) from Chapter 2.4.7. A system boundary represents the loading of the conveyor system, in this case a conveyor

tape. At the beginning, the component on the conveyor belt must be recognized, required for this a scanning device is used, which may use image processing and Quick Response (QR) scanning to fully identify the component and its position in space describes. Ideally, this identification process can be integrated into the conveyor unit integrate. Thus, the conveyor unit only had to provide the information about which position the robot should move to and which component it was acts. In general, the position has six degrees of freedom and can be determined with map to a struct. The component had to bring several pieces of information with it, but above all a gripping tactic and the gripper required for this. recognized as a human a suitable gripping tactic is intuitive, but this is very difficult with machines to realize and forms a separate research topic. In addition to the gripping tactics, that should Provide a unique ID and a mass for the component. The ID is used for identification in the logistic system and the mass can be used to after gripping the component to determine whether this was also carried out successfully became.

The mentally tracked component is now ready for collection. The for the Information required for collection is provided by the conveyor system and indirectly by the Component itself made available. If a control unit requires an assembly of components, the handling robotics is informed that the conveyor system for pick up the component provided. For this, the handling robotics becomes another given target position. The transition of the component begins in the direction of the process system or, in the meantime, in the direction of a tiling zone. The ones within the The information made available to the system is the component and its target position. In order to avoid malfunctions, e.g. B. a power failure, starting To facilitate the target position, this information should be stored persistently to avoid any loss of information and thus an uncontrolled avoid condition.

The target position is defined by a predefined placement layout or a policy determined, which are stored in the control process. The component is then direct or in a roundabout way in the process plant and waits for the process to start. The process is fully monitored by the plant and its parameters are recorded in a database if required. Then about the component ID creates a link to this entry in the database. After graduation of the process, the component is removed from the process system and either placed directly or with an intermediate step on a conveyor system for transport. Finally, all components should be registered again to ensure that the process could be carried out successfully. The component is now at the second limit of the system to be controlled.

## 3.2 Definition of Classes

Based on chap. 3.1, classes are now derived that can map the process. The class designations 'member' and 'component' are introduced here. The 'member' corresponds to an active participant of the process like the conveyor belt or the hardening plant. The term 'component' refers to a passive Component such as the part or the charging plate. The introduced attributes and methods are for documentation as UML class diagrams in the source code deposited. It should be noted here that a lot of effort went into the definition of the attributes and the methods, but was not included in the work for reasons of space. Instead, only the essential classes are listed here presented. The defined states can be seen in the appendix (12.3).

- Plant Component: The superclass, which contains attributes and methods, which are required by all classes. This includes e.g. B. the position and visualization.
- Plant: Representation of the hardening plant. Distinguished by variable loading places, which are determined by stack height and number of stacks will.
- Robot: The handling robotics that takes over the transport of components and charging plates.
- Conveyor: A conveyor belt that moves components in a straight line. There is a light barrier on the conveyor belt which, when activated, shuts down the conveyor belt turns off.
- Storage: The idea of this class originally came from an initialization problem. During initialization, the objects of the 'Table' class must be assigned to a spatial position. Furthermore, this class can be used to implement an intermediate store for charging plates.
- Rotator: A seventh axis commonly used in robotic systems comes. It can accommodate charging plates and a rotation around the high Realize axis.
- Intake: One of the system limits. This class brings components into the system a.
- Outtake: Realizes the second limit of the system and therefore removes components from the system.

– Table: The charging plates, which can accommodate components and for the ¨
handling are required.

– Part: The component that is initialized beyond the system limits and also deleted again. As mentioned at the beginning, in the course of this work between cylindrical components and components in the form of tetrominoes. For this purpose, a component ID is introduced for differentiation (Tab. 3.1).

Table 3.1: Component classes and their ID

| Component | Designation [35] | ID |
|---|---|---|
|  | cylinder | 1 |
|  | smash boy | 2 |
|  | Teewee | 3 |
|  | Rhode Island Z4 | |
|  | Cleveland | 5 |
|  | Hero | 6 |
|  | Orange Ricky 7 | |
|  | Blue Ricky | 8th |

## 3.3 Architectural Design

The structure of the overall application is developed in this section. Following the module for the system components, the user interface (UI), the assembly and the process specification is presented, followed by the module for communication. Finally, a UML diagram of the overall application is created from the individual drafts and a directory structure for programming is derived from this.

### 3.3.1 Module for the system components

To the in chap. In order to make the components presented in Section 3.2 as maintenance-friendly as possible, they are combined in a module 'simulation members'. Here, the 'Plant Component' class contains all the attributes and methods that also

are required by all other classes of plant components. The position of the respective coordinate systems can also be implemented in a separate class. A mechanism is required for the classes 'Plant', 'Storage' and 'Conveyor' which, when requested by an object of the class 'Robot', generates a

returns the pick-up point and then locks it until pick-up. These pick-up points are implemented with the 'Component Slot' class. The representation using
UML can be seen in Fig. 3.1.

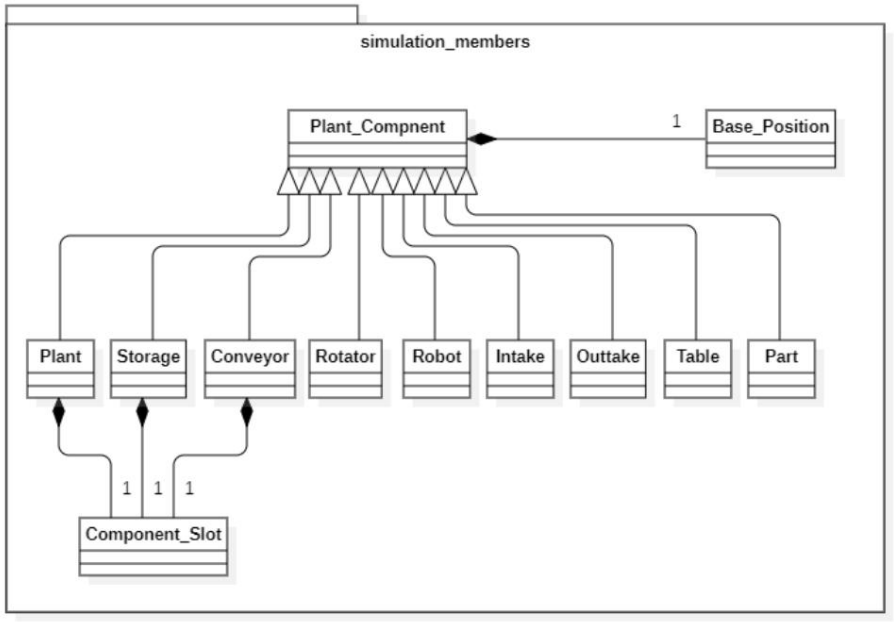



Figure 3.1: Module 'simulation members'

The diagram created in Staruml is exported as a fragment to create a
to be imported into other diagrams at a later point in time.

### 3.3.2 Module for the user interface

The Tkinter library, a standard library in Python, is used for the UI. For this purpose, a 'Simulation App' class is created, which is derived from the 'tkinter.Tk' class. Furthermore, the class 'Simulation App'

pages of the application to be displayed. These pages generated for this purpose are combined in a module 'simulation members' and when instantiating the
Application created. The combination in a separate module is intended to make a possible extension of the application more user-friendly. The classes 'Home Page', 'Draw Page', 'Settings Page' and 'Simulation Page' in the module are derived from the class 'tkinter.Frame' and enable the

Interacting with the objects from the 'simulation members' module. That for this
The UML diagram created is shown in Fig. 3.2.

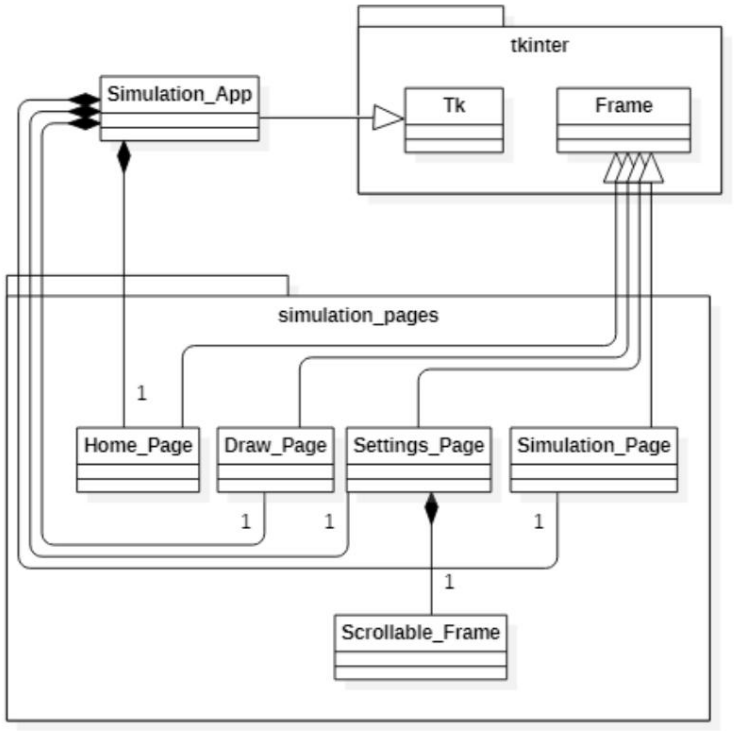



Figure 3.2: Module 'simulation pages'

### 3.3.3 Module for the tessellation rule

In order to be able to guide the process and determine the placement position of the components, regulations are needed that depend on the current state of the tessellation or the depend on the overall system. The regulations are built with the policies as they are in chap. 2.6.4.1 are identical. Here is also the core of this work integrated at a later point in time, namely as a 'charge policy', the tiling of various components. This 'Charge Policy' shall depend on the current load state, return a number of required actions, consisting of instructions for the three degrees of freedom: rotation angle of the 'rotator' and xy coordinates for the handling robotic arm. The Policy' class serves to control the overall process. Only one method is developed for this, in order to show that a control of the overall process by a complete description of the system is possible, which is realized by the defined states. The representation using UML can be seen in Fig. 3.3.

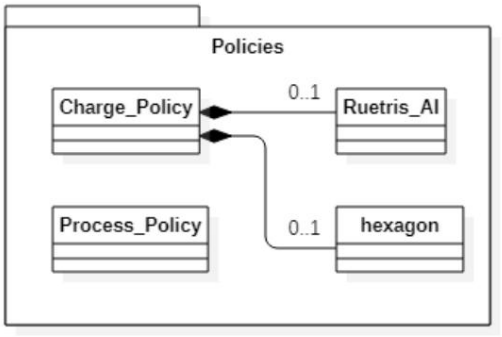



Figure 3.3: 'Policies' module

### 3.3.4 Module for the communication drivers

Communication drivers are required for communication between the created program and the real system components, which are also in a
own module 'communication'. Since there is currently only one training cell available as a test system, one driver is only provided for this purpose. The training cell, as in Chapter 3.5.1 described, consists a robot and a rotator. The structure of the communication is

Chapter 3.5.2 developed.

### 3.3.5 Structure of the overall application

In this section, the individual drafts are combined into an overall draft.
taken, the functionality of which is described as follows:
When starting the program, an object of the class 'Settings' should be created,
to load default values and to be able to change any values. In this object
objects of the classes noted as dataclasses are also created during runtime, which contain the standard values for the respective classes. Then the main class of the program is to be instantiated, for this purpose the object of the class 'Settings' is passed along with the superclass 'tkinter.Tk'. The object of the 'Simulation App' class then loads the 'simulation pages' module and instantiates it
one object each of the included classes. The object of the class 'Home Page'
manages the number of required components and therefore has a dependency
to the object of the 'Settings' class. The object of class 'Draw Page' is instantiated
based on the information from 'Home Page' stored in 'Settings',
the respective classes from the 'simulation members' module. using the object
of the 'Settings Page' class can access the parameters of the created member objects,

to put it more precisely: the affected dataclasses, accessed and any values
be changed. The changes made can then be used again in the object
of the 'Draw Page' class. To all parameters in the window below
to bring, another class 'Scrollable Frame' is instantiated, which houses the parameters. If
one is satisfied with the layout, the object of the class
'Simulation Page' can be used. The created layout is displayed in this window
loaded and the appropriate settings for the policies selected. Were the
settings is selected, an object of the corresponding class is created in each case.
Now the system can be animated. Furthermore, during the animation
there is the possibility of following the states of the system components.

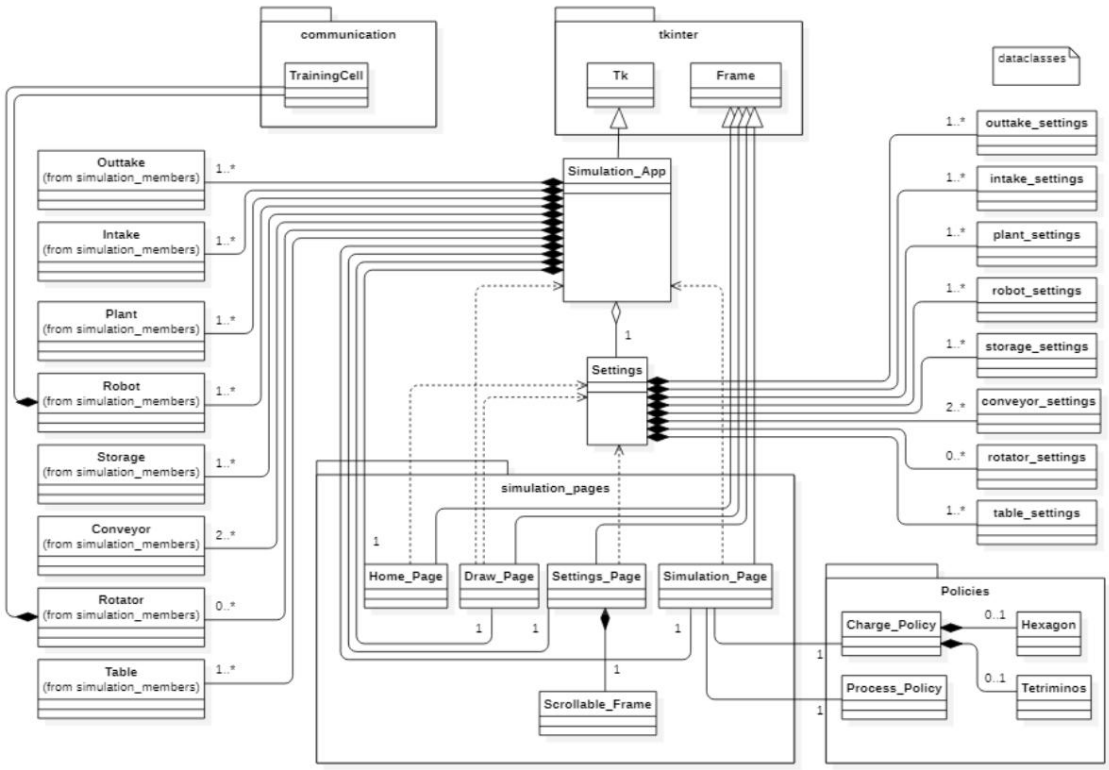



Figure 3.4: UML diagram of the overall application

It should also be mentioned that the 'simulation members' module was imported as a
fragment, so the module frame and dependencies are inside the
module not visible. This was chosen for reasons of clarity.

## 3.3.6 Code Generation

The charts were generated using the evaluation version of Staruml. This software offers
the possibility to create a code skeleton and a directory structure for

to create the project (Fig. 3.5). However, the software also creates its own files for the as dataclasses noted classes. However, this is a Python equivalent of a struct, so these files are deleted and moved to the File 'Setting.py' added. Furthermore, the main directory is still a File 'main.py' added and folder 'tkinter' deleted as it can be easily imported as an external library. The resulting program structure is shown in Fig. 3.6.

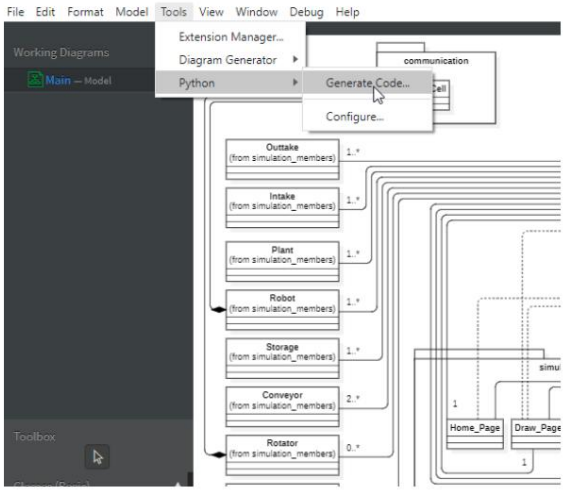



Figure 3.5: Code generation UML

```
C:\TEMP\Downloads\code_gen\Model>tree /F /A
Auflistung der Ordnerpfade für Volume OS
Volumeseriennummer : 8700-7F23
C:.
|   Settings.py
|   Simulation_App.py
|
+---communication
|       TrainingCell.py
|       __init__.py
|
+---Policies
|       Charge_Policy.py
|       Hexagon.py
|       Process_Policy.py
|       Tetriminos.py
|       __init__.py
|
+---simulation_members
|       Base_Position.py
|       Component_Slot.py
|       Conveyor.py
|       Intake.py
|       Outtake.py
|       Part.py
|       Plant.py
|       Plant_Compnent.py
|       Robot.py
|       Rotator.py
|       Storage.py
|       Table.py
|       __init__.py
|
\---simulation_pages
        Draw_Page.py
        Home_Page.py
        Scrollable_Frame.py
        Settings_Page.py
        Simulation_Page.py
        __init__.py
```

Figure 3.6: Directory structure

## 3.4 Tessellation Strategy

This chapter deals with the tiling of the charging plates. In principle, a distinction is made between two types of components and their tiling: cylindrical and tetromino-shaped components. It begins with the consideration of cylindrical components. On condition that only similar ones

If components are placed on a charging plate, this tiling can be described analytically. Then the tetromino-shaped components considered. Since no analytical connection can be created here, or at least this is not known, an AI is trained according to the RL method.
A corresponding environment is required for this and a program code for the workout.

### 3.4.1 Tiling of cylindrical components

The hexagon shape seems to be the most suitable for the tiling of cylindrical components (Section 2.5.2), so a mathematical description of the

Implementation wanted. The goal is a function that, due to the table diameter and
a characteristic length of the hexagon, positions and their order for
the occupancy is calculated. For this purpose, the tiling pattern is first drawn with Geo gebra (Fig. 3.7)
and possible, mathematically describable patterns
examined.

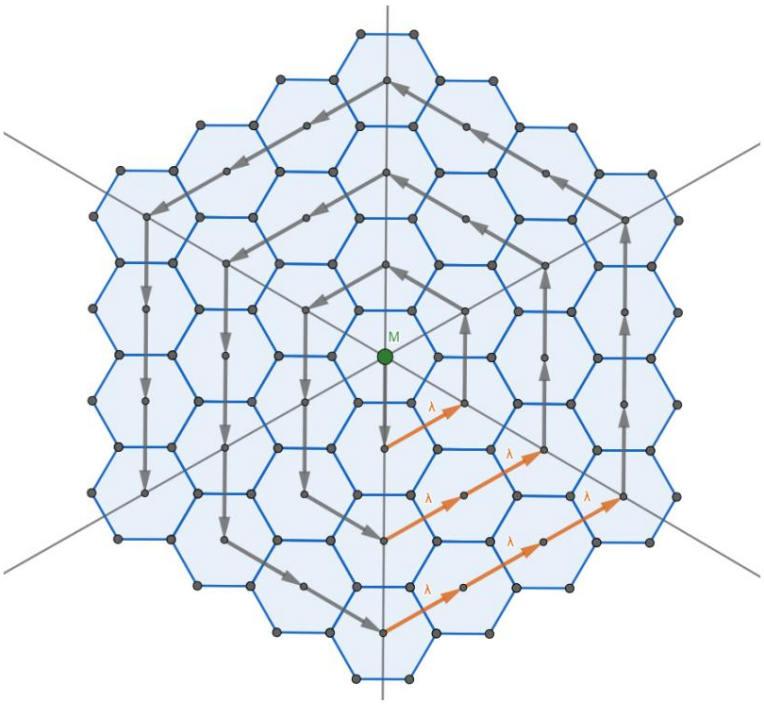



Figure 3.7: Hexagon tiling with Geogebra

It is noticeable that a connection of the midpoints is again in a hexagon shape
and the number of these points always increases by six towards the outside
(Fig. 3.9 and 3.10). If one now introduces direction vectors in order to clarify this property, one obtains
at the same time the suitable order for an occupation of the required positions. It is also noted that the
description of the centers can be expressed by a sum of direction vectors, the choice of which is limited
by the six different directions. Do you play it?

structure of the 'hexagon flower' in one's mind, it can also be stated that
that the number and direction of the vectors form a repeating pattern
(marked in orange). This information is enough to create a program code for the spiral structure of the ur
hexagons.
Program 3.1, 3.2, 3.3 and 3.4 show an implementation using Python, which
explained in the following paragraphs. It starts with the integration of the
required libraries.

Program 3.1: Libraries

```
import math
import numpy as np
import matplotlib . pyplot as plt
```

Three libraries are used for the prototype: Math around the angles function that are required for the description of the direction vectors. Numpy to be able to use the calculation rules for vectors.

Matplotlib to debug the program code and graphically display the results illustrate. Now the actual definition of the function can take place:

Program 3.2: Hexagon function

```
def calcHexagons ( hexInnerDia , tableDia ):

pointArray = []
p0 = np . array ([ 0,0 ])
point Array . append ( p0 )
6k=1 ;
i =0;
history_k = 0;
notFinished = True

while notFinished :
        while len( pointArray ) <= k*6 + history_k :
            for j in range ( k):
                direction = dirVector ( i- history_k ,k , hexInnerDia )
                pointArray . append ( pointArray [i] + direction )
                i += 1
        k += 1
        history_k = len( pointArray )-1

        direction = dirVector ( i- history_k ,k , hexInnerDia )
        radius = ( tableDia - hexInnerDia )/2
        if np . linalg . norm ( pointArray [i] + direction ) > radius :
            notFinished = False

return pointArray
```

Two transfer parameters are given to the 'calcHexagons()' function, the inside diameter of the hexagon as a characteristic dimension and the table diameter, ie the size of the circular area to be tiled. At first he will required return value of the function, namely the list of points described in the Cartesian coordinate system. Then the origin of the

coordinate system as the first point 'p0' and the list of points over- ¨
give. In the next step, run variables and a termination condition are created in order to make the program structure clearer. After creating these variables, the outermost loop is started. This loop is executed until the termination condition is met. The termination condition is checked in the underlying loop. She evaluates the situation of the next one

point to be generated and breaks off the program loop if it is outside the circular area. The second loop of the program generates the
Outer midpoints in an imaginary circumnavigation using the direction vectors. Finally, within this loop is the for-loop, which maps the increase in the direction vectors per cycle. To make the function clear ¨
to hold, the problem of choosing the direction vector becomes in another function described. For this, 3 parameters are passed: First, an index valid for the current circulation, which (for the assignment see Fig. 3.9, 3.10) and the number of outer hexagons (for the assignment see Fig. 3.9, 3.10) and the

Parameter k, which determines the current instance and 'hexInnerDia' which the characteristic dimension of the hexagon is defined.

Program 3.3: Help function

```
define dirVector (i ,k , hexInnerDia ):

    if i == 0:
        return np . array ([ hexInnerDia * math .cos (3* math .pi/2) ,
                    hexInnerDia * math .sin (3* math .pi/2)])
    if i != 0 and i // k == 0:
        return np . array ([ hexInnerDia * math .cos (11* math .pi/6) ,
                    hexInnerDia * math .sin (11* math .pi/6)])
    if i // k == 1:
        return np . array ([ hexInnerDia * math .cos( math .pi/6) ,
                    hexInnerDia * math .sin( math .pi/6)])
    if i // k == 2:
        return np . array ([ hexInnerDia * math .cos( math .pi/2) ,
                    hexInnerDia * math .sin( math .pi/2)])
    if i // k == 3:
        return np . array ([ hexInnerDia * math .cos (5* math .pi/6) ,
                    hexInnerDia * math .sin (5* math .pi/6)])
    if i // k == 4:
        return np . array ([ hexInnerDia * math .cos (7* math .pi/6) ,
                    hexInnerDia * math .sin (7* math .pi/6)])
    if i // k == 5:
        return np . array ([ hexInnerDia * math .cos (3* math .pi/2) ,
                    hexInnerDia * math .sin (3* math .pi/2)])
```

Essentially, the function 'dirVector()' is only a case distinction with non-trivial conditions. The case distinctions can be put into words as follows: If the current point is the first of the new instance (i==0), the direction vector in the negative y-direction. When the current point is not the first, but the integer division is equal to the instance number 'k' is zero, then the direction vector should be in 11pi/6-Direction to be returned. This procedure is analogous for the remaining conditions.

Program 3.4: Test routine

```
# test routine
hexagondia = 80
tableDia = 1000

# plot middle points of hexagons
a = np . array ( calcHexagons ( hexagonDia , tableDia ) )
#b = hexagon filter (a, hexagonDia , , y = a .T                tableDia )
_
pl. plot (x , y ,'*-')

#plot table
angles = np . linspace ( 0          ,   2 * np .pi ,          150 )
radius = tableDia /2
a = radius * np .cos( angle )
b = radius * np .sin( angle )
pl. plot (a ,b )

# plot boundary
radius = ( tableDia - hexagonDia )/2
a = radius * np .cos( angle )
b = radius * np .sin( angle )
pl. plot (a ,b )

pl. xlabel ('x-coordinate ')
pl . ylabel ('y-coordinate ')
pl. gca () . set_aspect ('equal ', adjustable ='box ')
pl. shows ()
```

Values for the inside diameter of the hexagons and the table diameter are created in the test routine. These values then become the programmed ones Passed the 'calcHexagons()' function and saved the result in the variable a. In order to get the usual formatting for a plot command, the matrix is transposed. Then the visualization, the points are marked with a '*'

and linked together in the order of generation. Furthermore, will
nor the circumference of the table and the circumference of the termination condition in
included the plot. The graphic is output with the command 'plt.show()'
(Fig. 3.8).

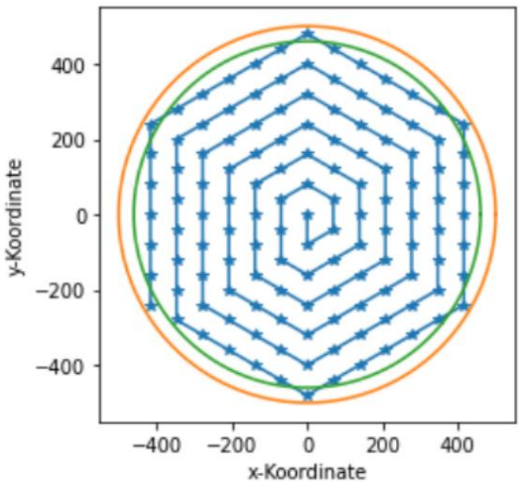



Figure 3.8: Result of the program code for tessellation

The result meets the required conditions. As indicated in the main routine (b commented out), filter functions still have to be developed to remove the 'hexagon flower' from too distant points and selected positions
to free. Filtering of selected points is needed to accommodate
To create bases for the charging racks stacked on top. The implementation of such a filter function is described in Chap. 3.4.5 presented.

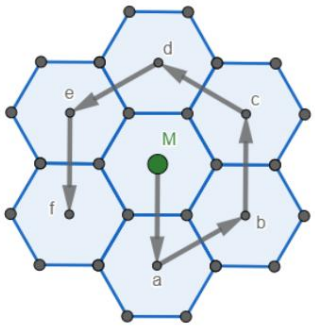



Figure 3.9: Hexagons 1st instance

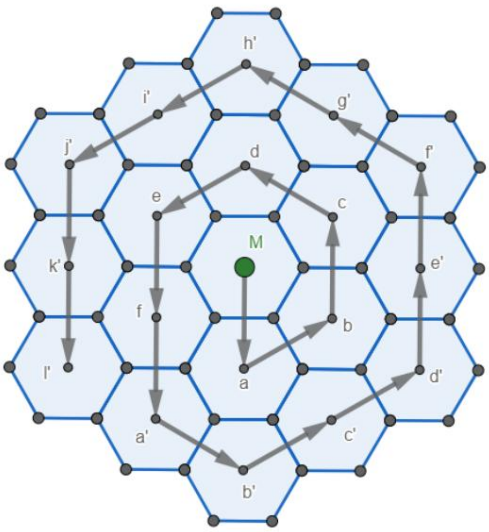



Figure 3.10: Hexagons 2nd instance

## 3.4.2 Tessellation of Tetrominos

Tetrominos [34] are geometric shapes composed of four squares, like
they are also known from the game Tetris. For the tiling of tetrominoes, the rectangular
form of the platonic tiling (Chap. 2.5.1) is used.
Again, a procedure is needed to find positions on a plane
to be able to generate automatically. For a possible implementation of the problem
is the use of an AI , according to the DQL and TDL method from Chap. 2.6.4.7 and
2.6.4.6, examined. For the stated purpose, an environment must be created in which the
AI can develop possible procedures. For this, first
developed a minimalistic prototype of the environment using Python, based on
which the following considerations were made.

### 3.4.2.1 Environment and Rewards

We begin with the discretization of the charging plate using a cuboid
Sub-areas as they emerge from the Platonic tiling (Chap. 2.5.1).
A characteristic length is required for this, which appears suitable for the approximation
of the components. Figure 3.11 shows this discretization of the charging plate and
components. The black area in the center represents a dead zone
which the components may not be placed. Such dead zones, i.e. areas
which must remain free in practice can be generated by means of suitable filters (Section
3.4.5), as is the case with the tiling of cylindrical components. Everyone
the cuboid represents an entry in a matrix that can be created with the Numpy library.
Criteria are required for the evaluation of discarded tetrominoes, which then allow points
to be awarded.
These criteria are elaborated in the following paragraphs.

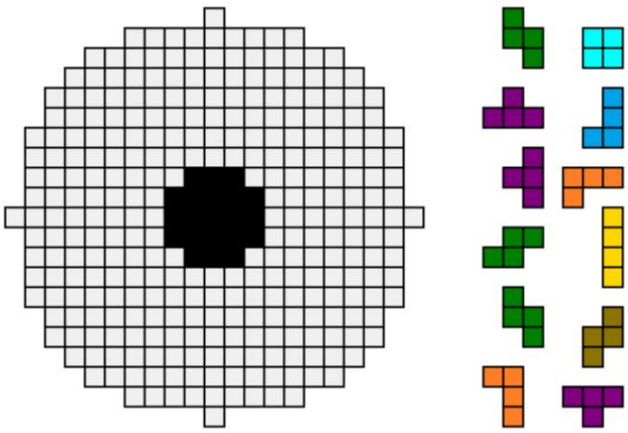


Figure 3.11: Discretization of components and charging plate

When charging, it is good to work from the inside outwards, so the distance from the charging plate to the centroid of the components appears as a suitable assessment measure. For this one determines the distances to each of the cuboid from the center of the charging plate and forms the mean value (Eq. 3.1, Fig. 3.12).

$$\tilde{x} = \frac{1}{4} X \tilde{x}i \tag{3.1}$$

By introducing deadzones, the beating of components or their placement within these zones will be assessed. However, the usefulness is of such an evaluation measure depends on the agent's options for action, since these possibilities can also be restricted, as is shown in Chap. 3.4.2.2 is implied. Figure 3.13 shows a violation of such a deadzone rule.

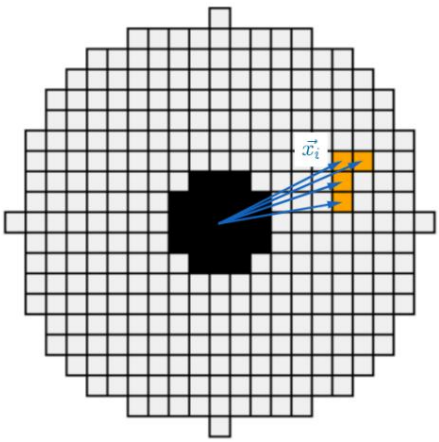


Figure 3.12: Centroid

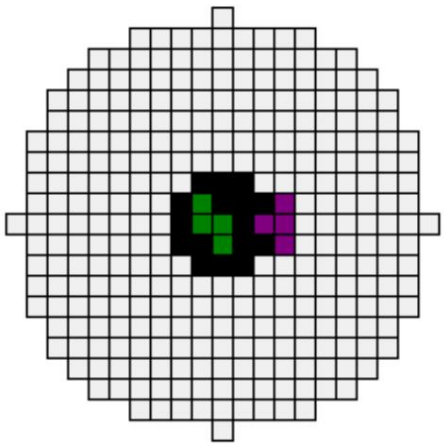
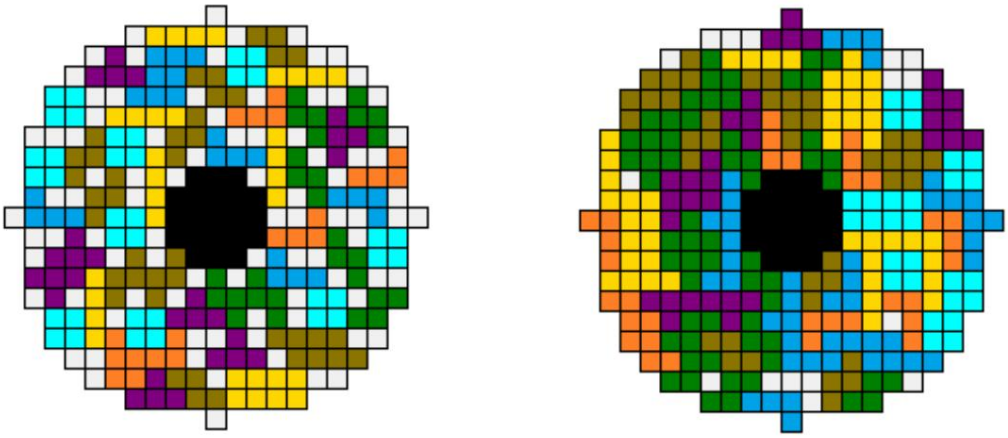


Figure 3.13: Deadzone rule

Another rating can be obtained by increasing a porosity value be won. About the term Pososity in relation to the evaluation criteria two representations (Fig. 3.14) were made. Figure a shows the High porosity tray whereas b shows low porosity tray.
In addition, b shows another property of the shelf: the unoccupied fields on the right top and middle below have the shape of a tetromino and would have been occupied could be if the appropriate component had appeared. Thus one arrives to assume that the type of distribution with which the components occur, a plays an essential role. A uniform distribution was used in this prototype, however, one could also implement other types of distribution.



(a) High porosity of the tray (b) Low porosity of the tray

Figure 3.14: Porosity of the deposit of tetrominoes

So far, the distance from the center ~x and the number of wasted places nwasted places have been found as potential evaluation criteria. One another possibility is counting the adjacent squares, therefore, it can be after the storage, which side edges have an immediate neighbor are counted. This results in a parameter nneighbours. What should be taken into account from here on is that all tetrominoes, except for Smashboy (Table 3.1), have 10 side edges. This therefore results in a disadvantage for this form.

$$Ri = \frac{\text{radius}}{|\sim xi|} \ \ddot{y}\ C1 \ \ddot{y}\ \text{nwasted\_places} \ \ddot{y}\ C2 + \text{nneighbors} \ \ddot{y}\ C3 + C4 \tag{3.2}$$

Equation 3.2 shows a possible reward strategy for placing a component. The constants C1 to C4 must be determined by experiments. However, one can now only evaluate the betting itself, so the question still stands after the possible actions of the agent open.

#### 3.4.2.2 Agent

In this section, two interaction schemes of the agent with the environment are presented. In the first variant one lets the agent from a pool of all possible placement positions, in the second variant from a pool of single actions. An action according to the first scheme is represented by a three-tuple shown. This tuple contains the desired x and y coordinates and the number of 90° rotations (Eq. 3.3, Fig. 3.15). Because this scheme of action the direct Predetermines storage at a target position, it is referred to as 'placement' in the following.

a = (x ÿ coordinate, y ÿ coordinate, n ÿ rotations) (3.3)

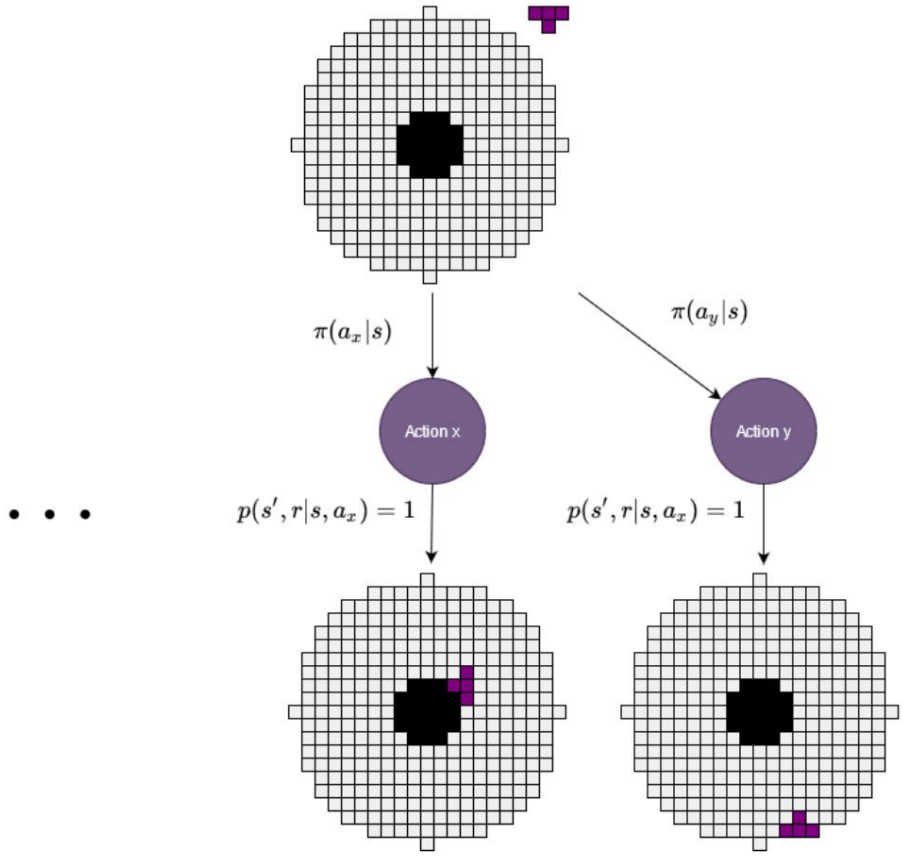



Figure 3.15: Action scheme: drop positions

The second action scheme is single-action control such as up, down, left, right, left turn, and right turn.
Figure 3.16 shows an interaction according to this scheme, which from now on will be called 'controller'. However, the individual actions must also have ¨
Points count towards the overall reward.
As shown in the illustrations, the transition dynamic usually plays no role in the schemes presented, since a certain action is always only in one of them
resulting state tastes. The biggest difference between the two schemes is the
Number of actions and the resulting number of states in a
episode can occur. However, the number of all possible states is the same,
namely 2n . The base of 2 stands for a binary coding (whether a field
is occupied or not) and the exponent n stands for the number of occupied fields (blocks of four). In this calculation one neglects the dependency
between the four squares, but the calculation depicts the situation well.

The prototype used here has 296 assignable fields, from which
the 2nd $\approx 4 \cdot 1.9e22$ possible states. This little calculation shows which
extensive problem to be solved.

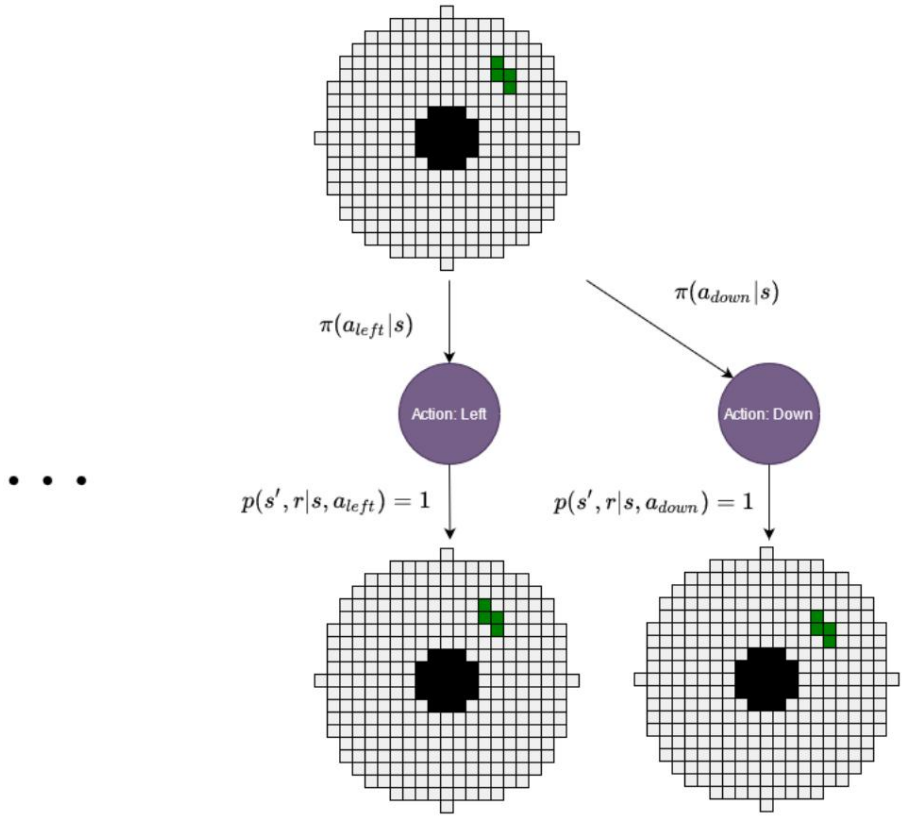



Figure 3.16: Action scheme: single actions

In the paragraph above it was mentioned that the transition dynamic mostly does not matter
plays. However, this is only correct with one exception: discarding parts. Namely, if components are
stored, only after this storage is a
new component determined. Figure 3.17 shows this branching of the transition.
Now the individual transition probabilities must also be described.
the. This can be done using a vector containing these probabilities (Eq. 3.4). This keeps the possibility
that
Transition probabilities must be specified depending on the problem, since a uniform distribution usually
does not depict reality with sufficient accuracy. Further thoughts on this can be found in Chap. 3.4.3.
Last but not least, it should be noted that also
the appearance position, which is always top left in Fig. 3.17, from random numbers
can be generated. This randomness could further influence the training.

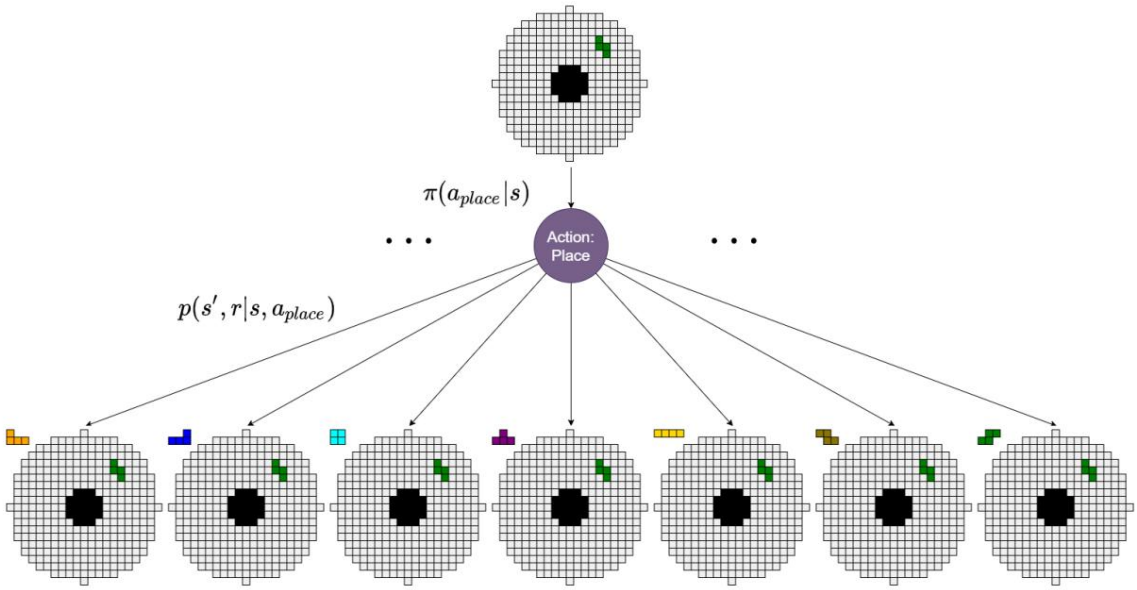
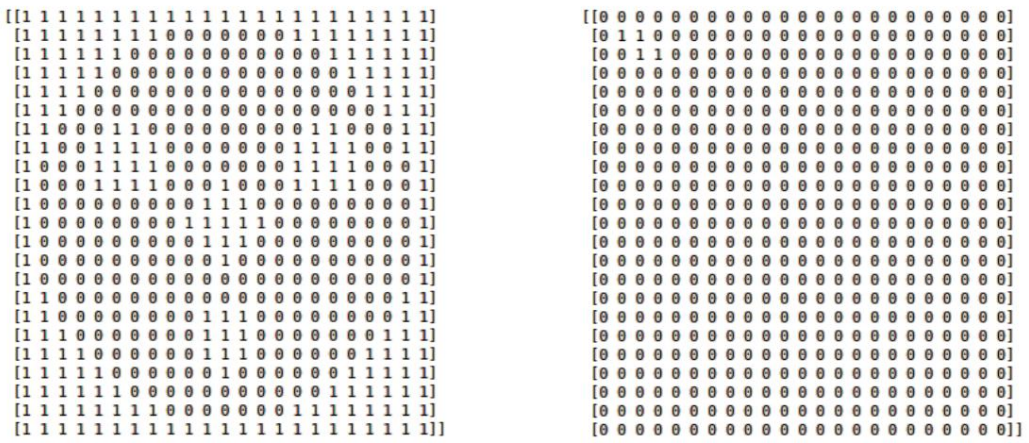



Figure 3.17: Single action: put component down

$$_{tp\ \_} = \text{h p0, p1, p2, p3, p4, p5, p6 i} \tag{3.4}$$

#### 3.4.2.3 Observation and Features

State representations, which serve as input for the neural network, must be designed for a learning routine (Section 2.6.1.7). A binary matrix notation is the first representation option. For this purpose, the environment is divided into two levels: one level represents the storage area, the other the movement of the component. Both levels are binary matrices of the same dimension. Figure 3.18 shows such a state representation for both matrices. This state representation is referred to below as 'full bool'.



(a) Matrix of the storage area

(b) Matrix of the component

Figure 3.18: State representation with binary values

A modified form of this representation is the generation of a single one
level from both levels. This has the advantage that the input vector in its
Dimension shrinks by half. Now, however, floating-point numbers for the
Requires representation because a Boolean description is no longer sufficient. container
If you add the numerical values for occupied and unoccupied with 0 and 1, numerical values must be
defined if the component is hovering over an occupied place or an unoccupied place. Since the beating
over an occupied place appears to be unfavorable, this is assigned the value -1. The beating over a place
appears favorable and is given the value 0.5. Thus the states would be on

limited to the value range [-1,1]. Figure 3.19 shows such a state
henceforth referred to as 'full float'.

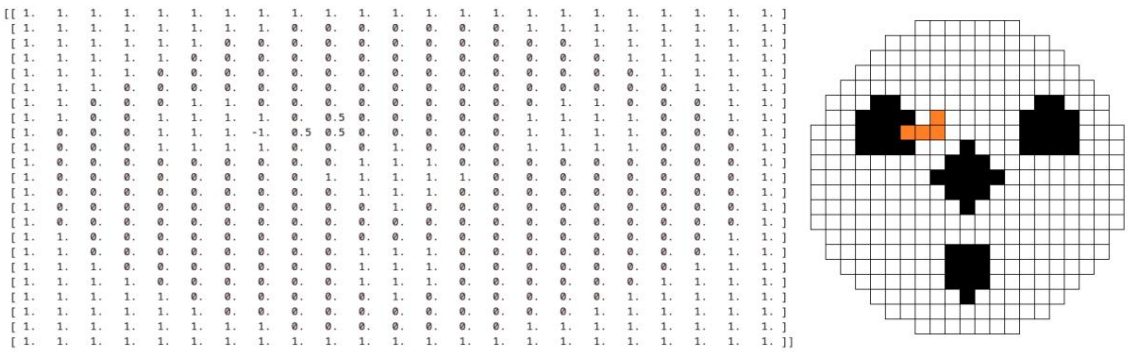
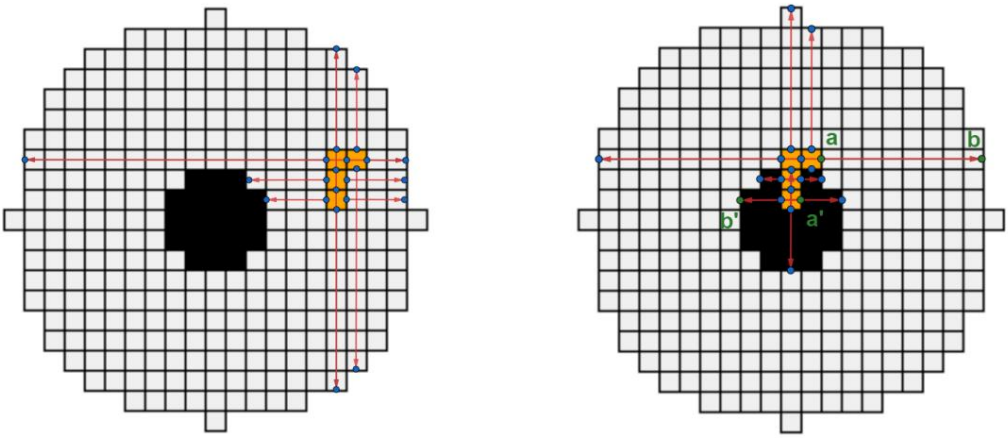



Figure 3.19: State representation with floating point numbers

A further possibility of a status representation is the view of the component
its surroundings, such as B. is practiced in self-driving cars. Therefor
the distance of all 4 side surfaces to a transition from dead zone to open space, or vice versa, is
calculated for all cuboids of the tetromino. If the component and the dead zone do not overlap, the count
is positive as shown in Fig. 3.20a. If
However, if there is an overlap, it is shown in the negative direction b is chooses. Figure 3.20
intended to clarify this difference. Considering the vector from point a on
At point b, the distance 8*block size is counted. On the other hand, consider the vector of point a
° on point b ° the distance -3*block size is counted. In addition
the centroids of the individual blocks can be added to this representation
will. All values must then be divided by a maximum value
can still be normalized to the value range [-1,1]. Because this condition description
contains the fewest values, it is henceforth referred to as 'reduced'.




(a) Positive measurement direction (b) Negative measurement direction

Figure 3.20: Measuring directions for the 'reduced' status display

#### 3.4.2.4 Neural Networks

This chapter deals with the decision-making process. For this
basically two different approaches are presented. The first approach deals with the selection of the action based on the estimated state-values,
while the second approach deals with the choice of action according to the DQL schema deals.

**Determination of state value**

Since the number of possible states after a transition is rather small,
all states resulting from the actions can be calculated in advance. For this follow-
states, the resulting state value can be determined via a neural network. The action is then selected by choosing the next state with the highest estimated state value. Figure 3.21 shows on the
on the left a set of possible consequent states that came about through the choice of the corresponding action. Sent through a neural network, the associated state values can be determined. It should be noted that these
Method for approximating the reward function (3.2) with ÿ = 0 is used
can be.

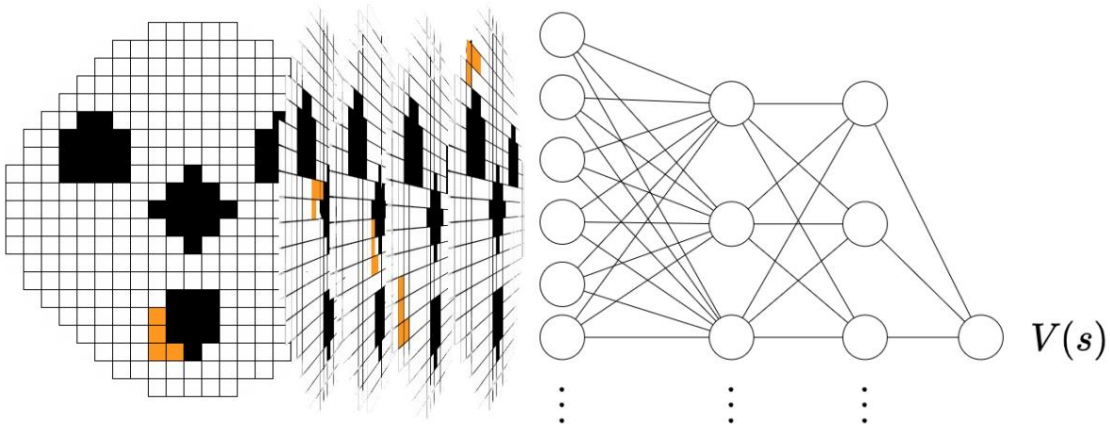



Figure 3.21: Estimator for ¨ state value

**Determination of q-value**

Since the number of possible actions in the action scheme 'controller' is manageable
are available, the procedure according to the DQL method is conceivable. For this ¨
the q-values for all m, possible actions estimated by a neural network (3.22),
which could be used to choose the best possible action. Here you can
the various network types presented in Chap. 2.6.4.7 use. One of the advantages of this
method is that not all possible subsequent states are determined
had to be. In addition, there would also be a movement from the start position to
End position also determined.

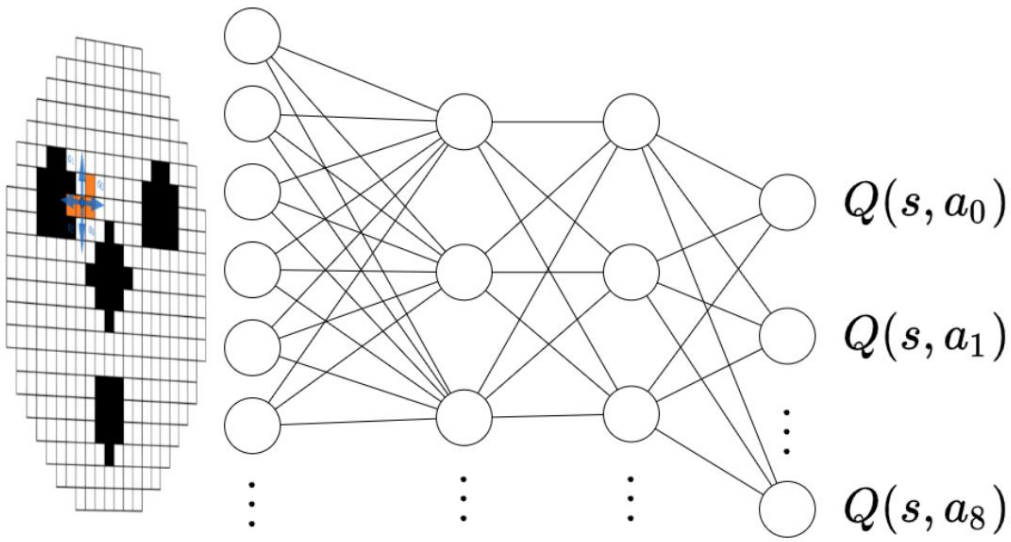



Figure 3.22: Estimator for ¨ q-value

#### 3.4.2.5 benchmarks

A method for evaluating the AI after the completion of the training routine is still missing. Therefore, a benchmark must be found that shows whether the AI performs well or poorly. For this, the constants of the reward function (Eq. 3.2) changed in experiments until the choice of the action with the highest reward, which can be determined for all possible storage positions, delivers a satisfactory result. If the computing time remains low, this method could also find their way into the created software. In addition, the success of this method is a prerequisite for all procedures presented in this chapter were to apply.

## 3.4.3 Determination of the transition probabilities

Chapter 3.4.2.2 pointed out the problem of transition dynamics, which occurs when components are stored (Fig. 3.17). For the description of this storage process, the transition probabilities were converted into a vector summarized (Eq. 3.4), which can be implemented in the environment. However, the question of finding the concrete probabilities is now open.
For this one could possibly include the components to be produced in a fiscal year, with the help of a classification AI (Section 2.6.3) in the respective component classes subdivide. If the components have been successfully classified, you can use a histogram use the superordinate distribution for the transition probabilities. Figure 3.23 shows two possible outcomes of such a process.
The probability distributions shown are, on the one hand, a uniform distribution (blue) and on the other hand a normal distribution (orange).
The transition probabilities must of course add up to 1.

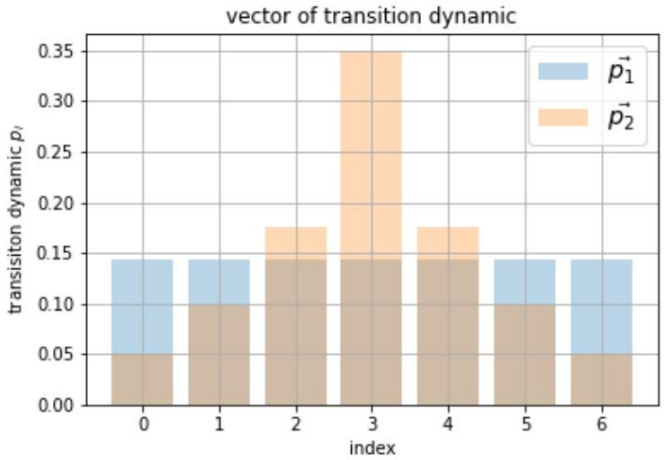



Figure 3.23: Plot transitions probability vector

## 3.4.4 Tiling by layout specification

Since there is not always a desire to determine the tessellation fully automatically, an interface for an external tessellation strategy is now to be developed. This is due to the fact that storage positions are often specified by expert process technology personnel. As an input format, it was decided in favor of the csv file, which contains the xy coordinates

the storage position includes. The workflow for the csv generation can be seen in Fig. 3.24. First, a charging plate is used in the company
Drawn by CAD program (Solidedge). Then a 2D development
made, through which the positions can be output as a drilling table. This drilling table can then be exported as a csv file via an Excel spreadsheet. In order to show that the additional seventh axis can also be used for process optimization, another one is used in the implementation

small optimization routine that minimizes the transition path from component pick-up to component deposit in two-dimensional space (Section 4.1.6).

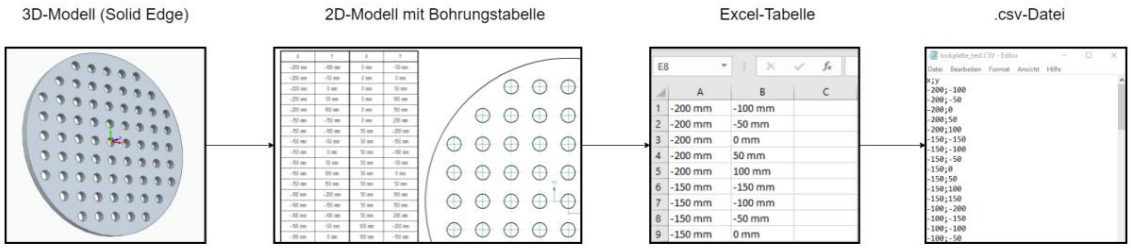



Figure 3.24: .csv generation workflow

## 3.4.5 Tessellation filter

Filters are used both for the hexagon pattern and for the environment of the AI
needed, which should ensure that certain areas are kept free. These filters should be structured with parameters in order to offer the greatest possible flexibility
to be able to. On the one hand, positions within an inner circle are to be filtered out in order to ensure gas circulation, on the other hand, open spaces are to be identified
needed for the legs of the parquet racks. For this one can, as shown in Fig. 3.25, use relative vectors ÿ from the respective centers of the dead zone
Form surfaces to the position itself. Now put these vectors in polar coordinates
data, a simple comparison operation between the first coordinate is sufficient
of the relative vectors and the half diameters 'middle dia' and 'dead dia' of the
deadzone areas.

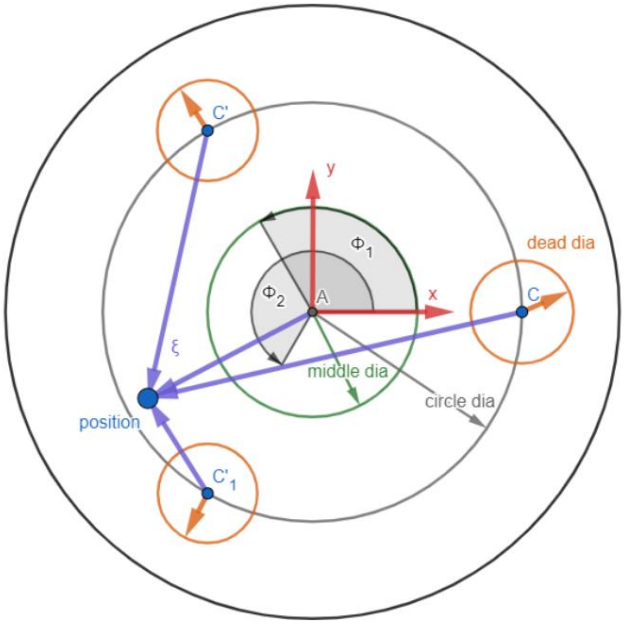



Figure 3.25: Filtering of positions

Program 3.5: tessellation filter

```
define filter_deadzone ( self , middle_dia , circle_dia , phis , dead_dia ):
# filter everything inside middle_dia - deadzone
for position in self . pointArray :
        polar = cart2pol ( position )
        if polar [0] < middle_dia /2:
            self . pointArray . remove ( position )


# filter everything inside standfoot - dead zones
for position in self . pointArray :
        for phi in phis :
            p0 = np . array ([ circle_dia /2, phi])
            p1 = cart2pol ( position )
            xi = p1 - p0
            if xi[0] <= dead_dia /2:
                pointArray . remove ( position )
```

## 3.5 Control of the training cell

So far, only concepts for the simulation and the charging process have been developed, but now a method is to be developed to test these concepts
to be able to apply to real systems. A server-client architecture is used for this

examined more closely. The server is an OPC UA server that
is created on a Siemens controller. This server should make certain parameters
accessible to the client in order to give them the option of
to overwrite parameters. As a proof of concept, a simple Pick&Place
Application implemented using a reduced version of the created
is program controlled.

### 3.5.1 Structure of the training cell

The heart of the training cell is a 6-arm robot from Kuka, the Agilus model. In the course of commissioning, a Simatic
S7 1500 installed by Siemens, using their Mxautomation functions
library can be used [36]. For an interaction with the PLC via
OPC UA initially uses a Raspberry Pi Model 4B to provide the necessary
create communication logic and pack it in a corresponding class,
which is then embedded in the simulation software.

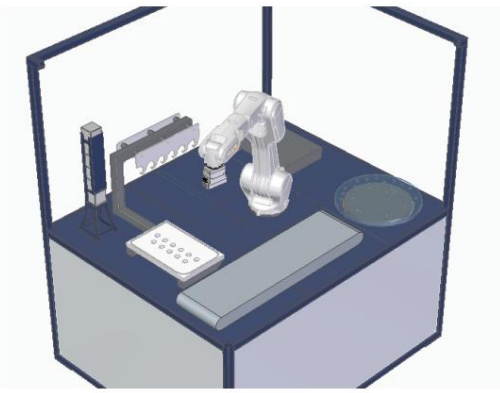


Figure 3.26: CAD model

Figure 3.27: Real model

### 3.5.2 OPC UA data model

Communication with the training cell takes place via a collection of variables on the PLC, which are made accessible via OPC UA. Siemens offers
Sufficient documentation and any examples for this [37] [38]. First will
a new data block is created in the project tree of the PLC. This includes the
required variables, which are then made available via the OPC UA settings. The definition of the interface was made and named by an external programmer. The term mass flow controller (MFC) was introduced as a designation for the software to be created. Table 3.2

shows the data block created for this, the substructures are described in Tables 12.3, 12.4, 12.5, 12.6 in the appendix.

Table 3.2: Data block: DB Interface MFC

| nodes | data type | description |
|---|---|---|
| MFC Alife | Boolean | Logical 1 if MFC online |
| MFC to Robot | Struct | data package: transmission from MFC to SPS |
| Robot to MFC | struct | data package: transmission from PLC to MFC |

### 3.5.3 Communication History

An essential point of communication is the 'heartbeat' to be created, which tells the PLC whether the driver is still running properly. For this purpose, the change interval of the intended bool value is measured. If the timer exceeds the five
exceeds the second mark, communication is assumed to have been interrupted. A handshake is also used in communication. Will
If data is sent by a participant, the corresponding bit: 'New DataValid' is set to logical 1 as the last step. The communication partner then copies
the value and sets the corresponding 'NewDataConfirmed' bit (Table 12.3, 12.4). Last but not least, both values are reset to logical 0, whereby the same order is observed. Figure 3.28 shows this procedure
using a rough sketch.

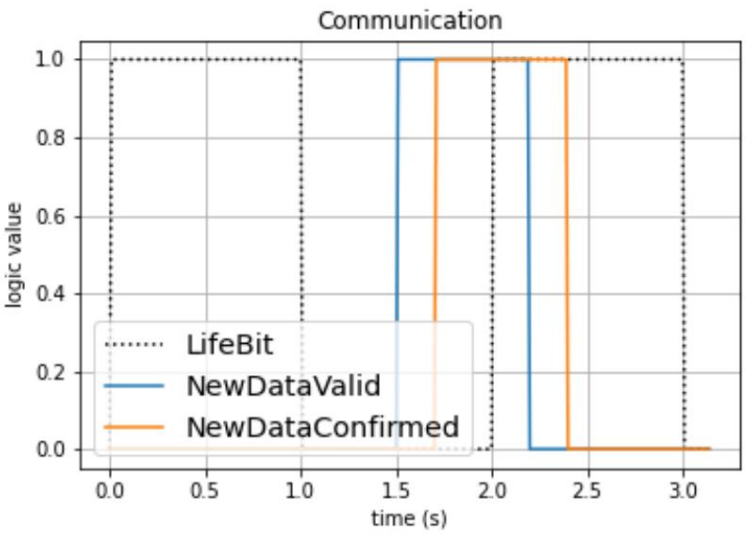



Figure 3.28: Communication scheme

The data is manipulated using the Python library Asyncua [39]. Program 3.6 shows what is probably the simplest example, the manipulation of a boolean value. Here, the UA TCP for unencrypted communication tion (Section 2.3.6). A subscription (Section 2.3.3) is created for the data exchange of the variables under the structure 'Robot to MFC' (Table 12.4).

Program 3.6: Standard NodeID Siemens S7 1500

```
from asyncua import client , etc

url = 'opc.tcp ://10.0.0.11:4840 '

# Connection setup using client
async with Client ( url=url ) as client :
# Read the desired node using the node ID
node_id = "ns=3;s=\"DB_Interface_MFC\".\"MFC_Alife\""
variables = clients . get_node ( node_id )
# Read current value
current_value = await variable . read_value ()
# Definition of a new, arbitrary value
new_value = ua . Variant (True , etc. VariantType . boolean )
new_data = ua . DataValue ( new_value )
# Set the new value
await variable . set_value ( new_data )
```

# 4 implementation

In this chapter, the implementation of the presented concepts (chapter 3) explained. After a short presentation of the simulation program, as well as their Adaptation for the training cell, the training of the AI is discussed. For this purpose, the points achieved per episode are recorded using the library TensorboardX recorded and visualized. For the software and hardware used, please refer to the appendix (Section 12.2).

## 4.1 Simulation program

It starts with the structure and a small functional description of the simulation program from Chap. 3.3.5. Then various configuration options for the layout generation are shown. A few implementations are then selected and described in more detail. For the documentation, the Pyreverse and Graphviz libraries were used to generate class diagrams that contain the attributes and functions that are used in the implementation

were created. These class diagrams are in the source code.

### 4.1.1 Structure of the simulation program

The simulation program consists of four windows, which correspond to objects of the four classes from the 'simulation pages' module (Section 3.3.1). Appears when starting first the home page (Fig. 4.1), in which the number of objects of the classes from the 'simulation members' module to be initialized is defined. In addition one defines the simulation area or its dimensions. The only one of the home page is the draw page (Fig. 4.2). On this site can by printing the draw buttons the initialization of the previously specified objects begin. A routine is triggered by clicking in the gray area and an additional window appears, which allows changing the class parameters. After completing the settings, the corresponding object in the Instance of 'Simulation App' created.
If you want to change any settings, you can switch to the settings page (Fig. 4.3). This page contains all parameters of the created objects that are required for the process handling are of interest. If you are satisfied with the layout, you can click the simulation page (Fig. 4.4) can be changed.

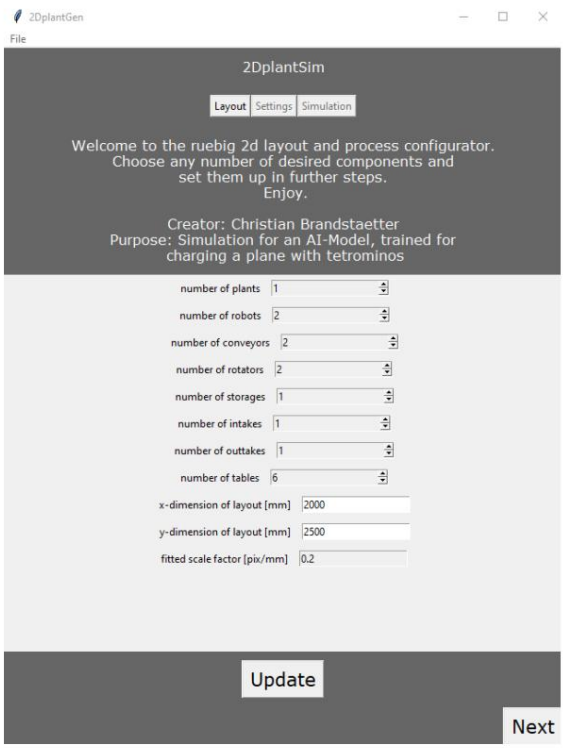



Figure 4.1: Home page

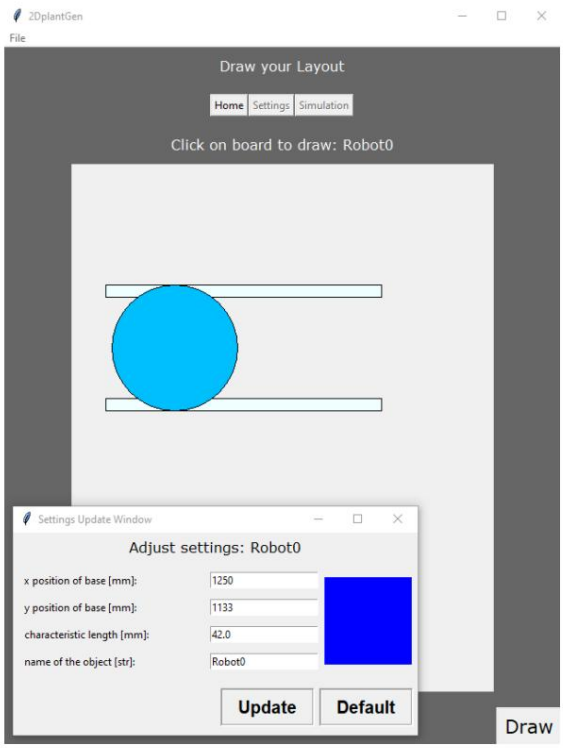



Figure 4.2: Draw page

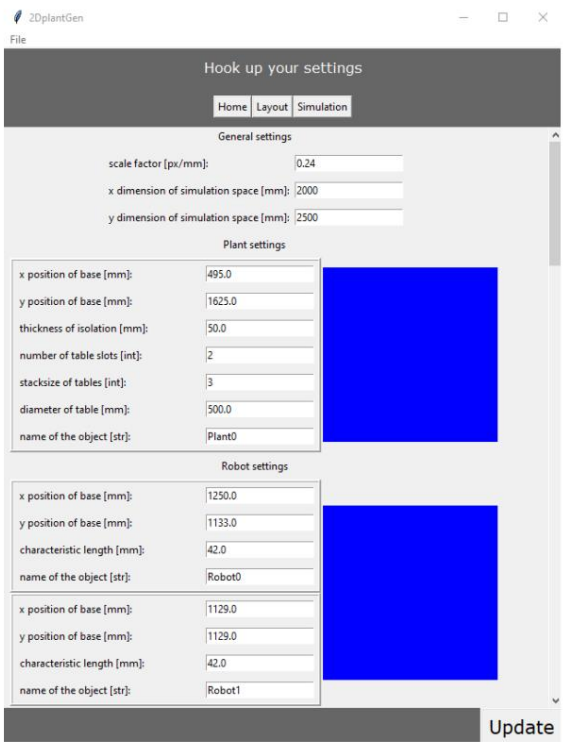



Figure 4.3: Settings page

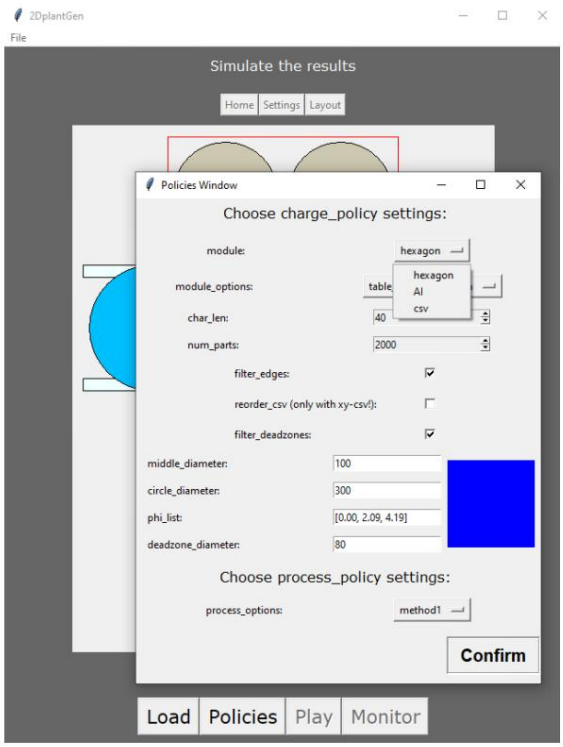



Figure 4.4: Simulation page

The layout is first loaded on this page, then the policies that control the behavior of the system can be set. In the window, Po

licies', among other things, the parameters for the tessellation filter are also entered. The system can then be simulated by pressing the play button
will. Furthermore, a monitor can be called up during the simulation,
which shows the current states of the individual system components.

### 4.1.2 Layout Configuration

Now we should turn to the configuration options and the scalability of the
systems are entered. Just like the tiling filter from Chap. 3.4.5 bring about a change in the level to be tiled in order to set the AI to changeable
environments, the layout creation should also be a certain
provide a degree of variability. This would be of interest if the topic were to be continued, since an AI can also be used to control an entire production facility
could develop. At this point, however, it should be noted that the entire program would have to be reconsidered and rewritten for this, since this implementation is at best to be seen as a suggestion.

Figure 4.5 shows a configuration with a tandem hardening plant with two places and a stack height of three. Furthermore, there are six tiling frames, two conveyor belts, a storage area and two rotators in use. The figure shows an unloading and a loading process. The second layout (Fig. 4.6) shows a configuration with a tandem hardening plant with three places and a stack height of six. Furthermore are
here six rotators, four conveyor belts and 18 tiling frames are in use. here
there are inlets on the middle conveyors and outlets on the outer
better. Both configurations are created with the 'process option: method1' (Fig. 4.4) controlled.

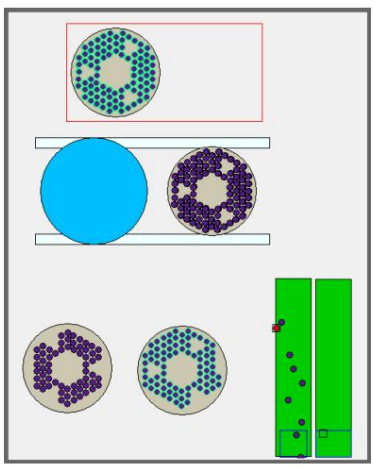


Figure 4.5: Layout 1

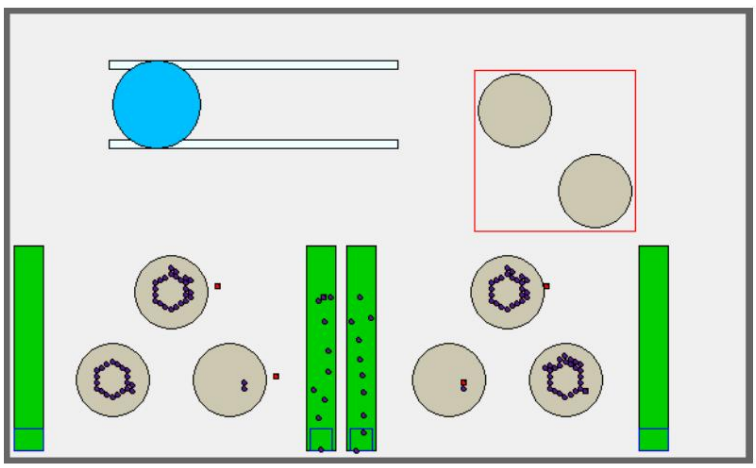


Figure 4.6: Layout 2

### 4.1.3 Linear and rotary movement

A speed is transferred to the object for the movements, and a new position of the object results from the integration of this speed over time (more precisely the call cycles) (Eq. 4.1). The new position is then discretized again and converted to pixel coordinates. The update of the objects
in the simulation environment happens with the difference of the time step in pixels. It should be noted here that no hard real time was implemented for the calls, the movement speed fluctuates depending on the call interval, given through the operating system. Figure 4.7 illustrates this procedure for the movement from A to B.

$$v(t) = \frac{dx}{dt} \quad =\ddot{y}\ x(t) = x0 + Z_{t0}^{t1} v(t)dt \tag{4.1}$$

Program 4.1: Linear movement

```
1  def move (self , x_layout , y_layout ,                    velo_max =8, epsilon = 0.1) :
2  # define lin.path :
3  dir_Vec = np . array ([ x_layout - self . x_layout ,
4                                        y_layout - self . y_layout ])
5  dir_Vec_norm = np . linalg . norm ( dir_Vec ) # length
6      if dir_Vec_norm <= epsilon : # reached ?
7          return True
8  elif dir_Vec_norm < velo_max : # close ?
9          self . dx = dir_Vec [0]
10         self . dy = dir_Vec [1]
11 else :
12         dir_Vec_unit = dir_Vec / dir_Vec_norm
13         velo_Vec = velo_max * dir_Vec_unit
14         self . dx = velo_Vec [0]
15         self . dy = velo_Vec [1]
16
17 self . x_layout += self . dx
18 self . y_layout += self . dy
19 self . coord_trans()
20 self . dx_pixel = int( self . scale * self . x_tkinter ) - self . x_pixels
21 self . dy_pixel = int( self . scale * self . y_tkinter ) - self . y_pixel
22 self . canvas . move ( self .id                    ,    self . dx_pixels ,          self . dy_pixel )
23 self . x_pixels += self . dx_pixels
24 self . y_pixels += self . dy_pixel
25
26 return False
```

Similar to the linear movement, the rotary movement is also implemented via the appropriate physical differential equation (Eq. 4.2). Discrete values for ¨ ÿ are given for the position update, but the Tkinter library allows this
only linear position updates, so the circle is approximated by a polygon (Fig. 4.8).

$$\ddot{y}(t) = \frac{d\dot{y}}{dt} \quad =\ddot{y}\ \dot{y}(t) = \dot{y}0 + Z_{t0}^{t1} \ddot{y}(t)dt \tag{4.2}$$

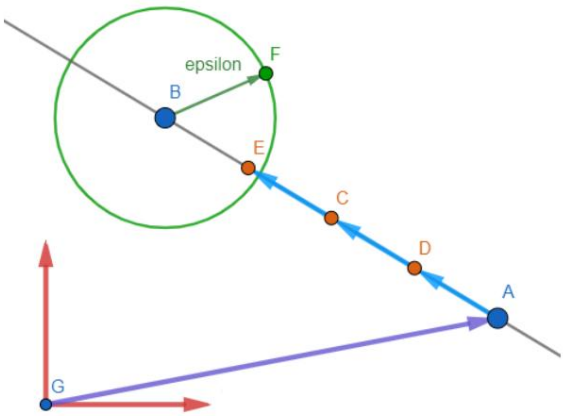



Figure 4.7: Linear motion

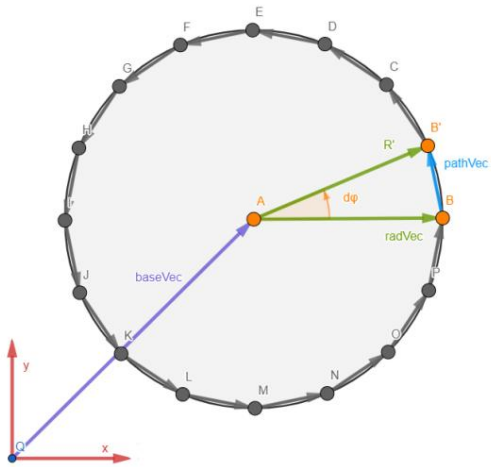



Figure 4.8: Rotation

Program 4.2: Rotatory movement

```
1 def rotate (self , phi , 2 # Rotation      n=128    , epsilon = 0.01) :
as polygon (n) => angle
3 d_phi = np .pi/n
4 phi_rel = phi- self . phi
5       if np .abs( phi_rel ) <= epsilon :
6           return True # reached ?
7 elif np .abs( phi_rel ) < d_phi :
            delta_phi = phi_rel # close ?
9 elif np .abs( phi_rel ) >= d_phi and phi_rel < 0:
10          delta_phi = - d_phi
11 else :
12          delta_phi = d_phi
13
14 for part in self . list_of_parts :
15          partVec = np . array ([ part . x_layout , part . y_layout ])
16          baseVec = np . array ([ self . x_layout , self . y_layout ])
17          radVec = partVec - baseVec
18          r = np . linalg . norm ( radVec )
```

```
        phi = np . arctan2 ( radVec [1], radVec [0])
        pathVec = r*np . array ([np .cos( phi+ delta_phi )                    ,
                                                np .sin( phi+ delta_phi )])- radVec
        part . dx += pathVec [0]
        part . dy += pathVec [1]
        # << position update like before > >
self . phi += delta_phi
return False
```

### 4.1.4 Subscriptions

Since several objects of the 'Robot' class can access the same subcomponent of the can access the entire system, e.g. For example, to pick up components, a method had to be implemented to manage this. For this purpose, the process is explained using the 'Conveyor' class, which contains such a method. The method will called by an object of class 'Robot'. First it is checked whether this instance has already placed an order, if not, will be in the second step assigned an order to this. A distinction is made here as to whether the object want to pick up or drop something off. Will subsequently be a method for the Collection or task called, the object (the customer) is removed from the Allocation list removed.

Program 4.3: Subscriptions

```
define reserve_slot_for ( self                    ,    intend , component_type ,                    subscriber ):
    if subscriber in self . subscriptions . keys ():
        return self . subscriptions [ subscriber ]

elif intend == 'put ' and component_type == 'part ':
        slot = self . get_slot_to_put_for ( component_type )
        if slot is None :
            print ( self . name            ,    ': no more slots available (put)')
            return
        self . update_state ()
        self . subscriptions [ subscriber ] = slot
        return slot

elif intend == 'take ' and component_type == 'part ':
        slot = self . get_slot_to_take_from ( component_type )
        if slot is None :
            print ( self . name            ,    ': no more slots available ( take )')
            return
        self . update_state ()
        self . subscriptions [ subscriber ] = slot
```

```
21          return slot
22 else :
23          raise Exception ('Invalid Intend ')
```

### 4.1.5 Complex rotation

Now the rotation by means of complex numbers is to be shown. For this one can use the exponential representation of complex numbers (Eq. 4.3) in order to (Eq. thus represent the simple multiplication to k 4.4). A sketch for the rotation in the complex plane is shown in Fig. 4.9.
This is a particularly elegant and compact approach.

$$\overline{z} = a + ib = |z|e^{i\arg(z)} \tag{4.3}$$

$$z1\ z2 = |z1\ ||z2\ |e\ i\ddot{y}(\arg(z1)+\arg(z2)) \tag{4.4}$$

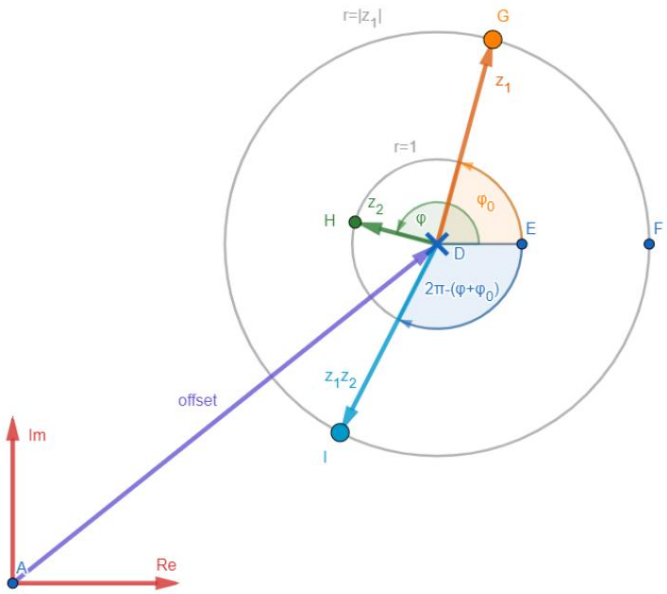



Figure 4.9: Rotation using complex numbers

Program 4.4: Complex rotation

```
1 def rotate_part (self , phi ): 2 z2 = math .e **
(-phi*1 j) # gorgeous 3 self . update_points () 4 offset =
complex ( self . x_pixel , 5 new_points = []
                                                self . y_pixel )
```

```
for x ,y in self . point :
        z1 = ( complex (x ,y)- offset )
        v = z2*z1 + offset
        new_points . append (v . real )
        new_points . append (v . imag )
self . canvas . coords ( self .id                    ,   * new_points )
self . canvas . tag_raise ( self .id)
```

### 4.1.6 Two-dimensional path optimization

In order to keep the travel distances within the simulation (and partly also on the real system) small, a small optimization of the transition movement is carried out for performed. For this purpose (Fig. 4.10) the position of the distance vector is simulated and

taken, which by the procedure from chap. 3.4.4 was won. the
Coordinates of the csv file are related to the base (v,w), as in Fig. 4.11
will be shown. To solve the minimization problem of the form: minÿ | ~d| to formulate, will
the absolute value of the distance vector determines:

$$\sim xg(ÿ) = \sim xb + ÿ\sim(ÿ)$$

$$\sim d(ÿ) = \sim xs\ ÿ\ \sim xg(ÿ) = \sim xs\ ÿ\ \sim xb\ ÿ\ ÿ\sim(ÿ)$$

$$| \sim d(ÿ)| = q\ (xs\ ÿ\ xb\ ÿ\ |ÿ\sim|\cos(ÿ))2 + (ys\ ÿ\ yb\ ÿ\ |ÿ\sim|\sin(ÿ))2$$

The task was solved using the Python library Scipy. The created program code can be seen in Prog. 4.5. This routine returns a policy as used in the simulation program.

Program 4.5: path minimization

```
import numpy as np
from scipy import optimize

pointArray = np . loadtxt ('policies/ data.csv ', delimiter =';',
                                        skiprows =1)

def policy ( ** kwargs ): # state as dictionary
        idx = kwargs [' parts_on_table ']
        pos = pointArray [idx]
        vi = pos[0]
        wi = pos[1]
        beta = np . arctan2 (wi                ,   vi )
```

```
13        psi = np . sqrt (v ** 2 + w ** 2)
14        xs = kwargs [' x_layout_robot ']
15        ys = kwargs [' y_layout_robot ']
16        xb = kwargs ['x_layout_table ']
17        yb = kwargs ['y_layout_table ']
18        def F ( phi ):
19            return ( x0-xb-psi*np .cos( phi )) ** 2 + \
20                        ( y0-yb-psi*np .sin( phi )) ** 2
21
22        res = optimize . minimize_scalar(F                    ,  bounds =(-np .pi , np .pi) ,
23                                                                    method ='bounded ')
24        phi = res . x
25        x = psi*np .cos( phi )
26        y = psi*np .sin( phi )
27
28        is_last = True if idx == len( pointArray )-1 else False
29        return {'phi ': [ beta -phi], 'x': [x],
30                    'y': [y], 'is_last ': is_last }
```

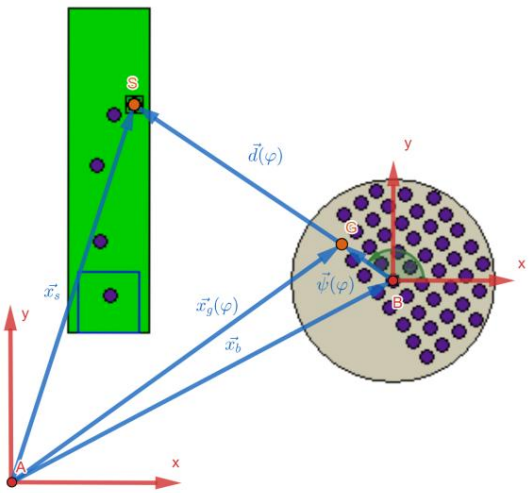



Figure 4.10: Path vector

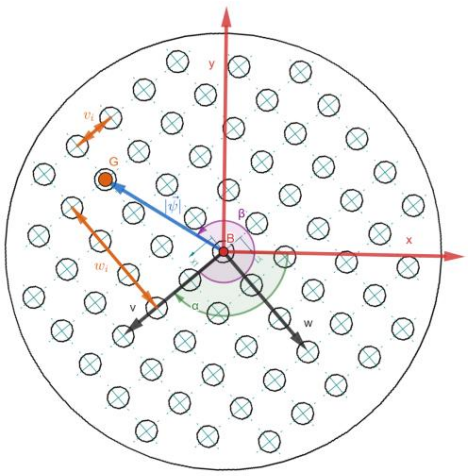



Figure 4.11: Coordinate specification

## 4.2 Adaptation to training cell

A modified form of the simulation program is also to be created, which uses the communication driver created in Chap. 3.5.1 includes. The program
After successful adaptation to the training cell, is loaded onto a virtual machine (VM) (similar to the development environment), which receives access to the network card of the PLC via a virtual network. Thus, after successful

Testing the program can be used at any time to test its functionalities
to demonstrate without having to deal with the environment setup. In the context of the automation pyramid
(Section 2.1), the program is in
the process control level.

### 4.2.1 Layout

All 'simulation members' except for one 'conveyor', one 'rotator', one
Removed 'Table' and one 'Robot'. The arrangement is given by Fig. 4.12.
Since components are neither brought into the system nor removed, the use of system boundaries is
obsolete. Therefore a new 'process policy' is created,
which alternately transfers the components between B1 and B2. The transfer will
inverted if the entire layout is occupied, or no more components in memory
available.

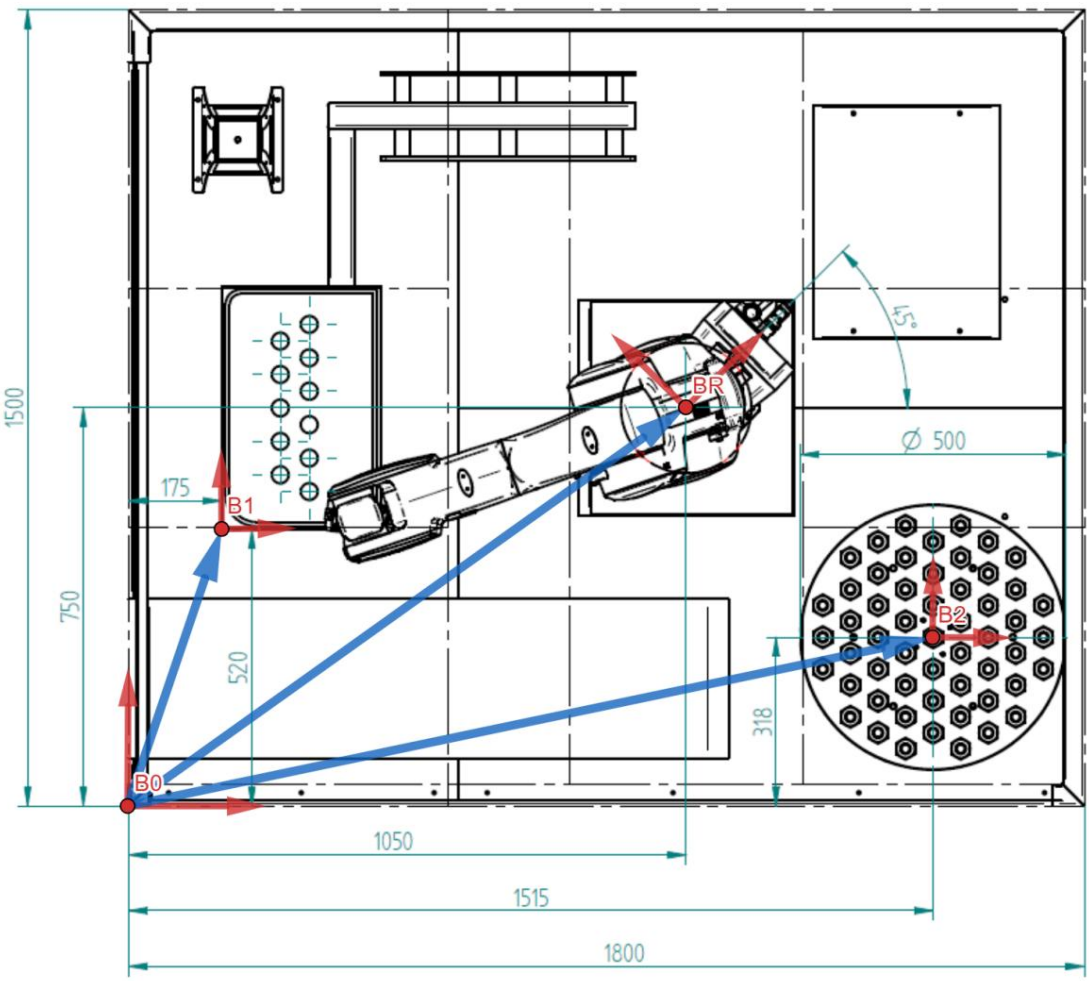



Figure 4.12: Training cell layout configuration

### 4.2.2 Simulation and Control

The training cell is controlled via the new 'process policy',
for this purpose the driver f Only the communication can be switched on (Fig. 4.14). This activation should only take place after a successful simulation.
After initialization, the cylindrical components are in the intended positions (Fig. 4.13). Since the real system is very fast at 100% overdrive
acts, the simulation speed was increased to 100 pixels per call cycle (Prog. 4.1) and the divider for the angle increment of the rotation was reduced to 16 (Prog. 4.2). A video for the control is in a YouTube playlist [40]. The large delays result on the one hand from the
Data transmission and on the other hand by a function block for the coordinate conversion.

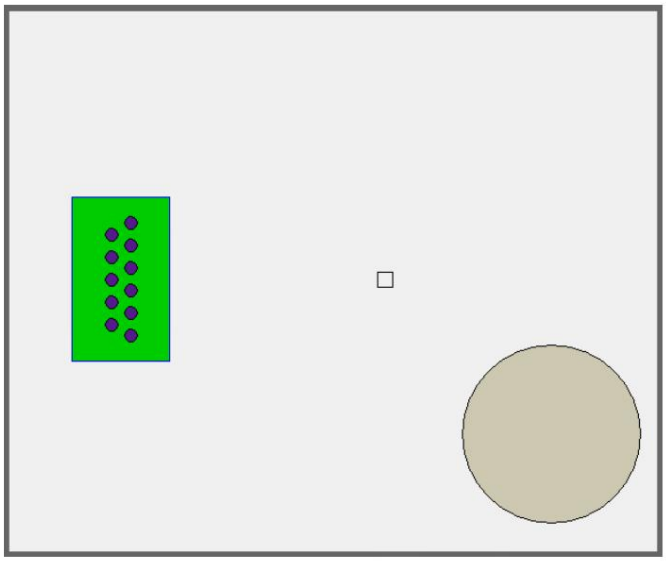


Figure 4.13: Simulation layout

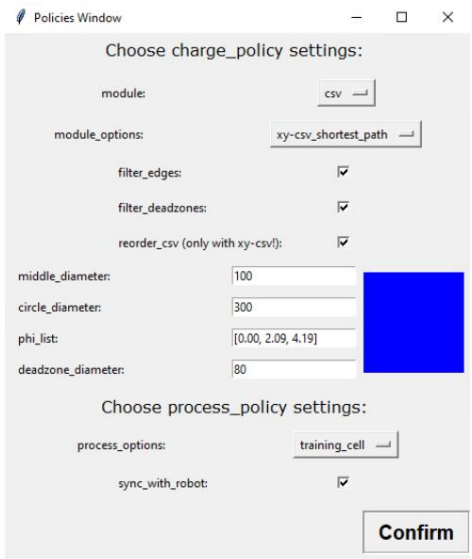



Figure 4.14: Policies

## 4.3 AI Implementation

The code for the agent, the evironment and the networks as well as the training and Test routines were developed in Python. All source code is on Github [41] freely accessible, so the implementation of the environment and the agent not further discussed here. It should be noted here that the structure of the code with examples from the Pytorch website and open-source projects oriented [42] [43] [44]. In this chapter, however, the training routine, and the neural networks, which are essential for learning success. In the next

Chapter the results are presented depending on the selected parameters
and interpreted.

### 4.3.1 Learning Routine

The structure of the learning routine will now be considered. At the beginning, the required
Libraries loaded (Prog. 4.6). The 'Controller Agent' implements the corresponding de, in
Chap. 3.4.2.2 presented method. The 'Placement Agent' the first method
of the same chapter. The 'Benchmark Agent' (Section 3.4.2.5) is an agent without a new
ronal network and was used for the fine tuning of 3.2) and for the reward function (Eq.
automatic generation of (s, s', r, d)-tuples for the application of SL.

Program 4.6: Training Routine: Libraries

```
1 import argparse
2 import json
3 from tensorboardX import SummaryWriter
4 from torch info import summary
5 from src . agent import Controller_Agent
6 from src . agent import Placement_Agent
7 from src . agent import Benchmark_Agent
8 from src . environment import Ruetris
```

Next, the parameters to be set and hyperparameters are required.
These are created using a function (Prog. 4.7). To save space
short, the help texts of the parameters were deleted; in this regard, reference is made to the
source code [41]. Most of the designations should be self-explanatory,
some will now be addressed. The '–state report' parameter determines the type of
representation that is transferred to the neural network. These are the ones in Chap. 3.4.2.3
presented state representations. '–action method' determines, in addition to '–reward action
dict', the agent to be used, which is specified
the corresponding network type (chap. 3.4.2.4) is used with the parameter '–net type'. Only
two-layer networks are used throughout the experiment
However, the number of neurons in these layers can be set with the parameters '–statescale'
to be changed. The rest of the parameters are basically the same
around the hyperparameters presented in chapter 2.6.

Program 4.7: Training Routine: Arguments

```
1 define get_args ():
2 p = argparse . ArgumentParser(
3     ''' Implementation of RL - Learning - Agent playing Ruetris ''')
```

## 4 implementation

```
# environment variables :
    p. add_argument ('-- table_dia ', type =int                    , default = 420)
p. add_argument ('-- block_size ', type =int                    , default =20)
    p. add_argument ('-- state_report ', type =str p. add_argument ('--      , default = 'full_float ')
    action_method ', type =str                                   , default = 'placement ')
p. add_argument ('-- rnd_pos ', type =bool 10 p. add_argument ('--   , default = False )
rnd_rot ', type =bool                                            , default = False )
    p. add_argument ('-- reward_action_dict ', type =bool p. add_argument ('--c1 ',      , default = False )
    type =int default =2)                                        ,
    p. add_argument ('--c2 ', type =int p.add_argument           , default =50)
    ('--c3 ', type =int p.add_argument ('--c4 ', type =int       , default =3)
                                                                 , default =5)
# agent and network variables
p. add_argument ('-- net_type ', type =str 18 p.add_argument ('--    , default = 'dq-net ')
statescale_l1 ', type =int p.add_argument ('-- statescale_l2 ', type =int          , default =10)
    p.add_argument ('-- batch_size ', type = int p.add_argument ('--lr ', type    , default =6)
    =float p.add_argument ('--lr_decay ', type =float 23 p.add_argument ('--default = 128)
    final_lr_frac ', type =float 24 p.add_argument ('--gamma ', type =float default = 0.01)
    default = 0.2)                                               , default = 1.0)
                                                                 , default = 0.001)
                                                                 ,
    p. add_argument ('-- initial_epsilon ', type =float p. add_argument ('--      , default =1)
    final_epsilon ', type =float 27 p. add_argument ('-- epsilon_decay ', type =float default = 0.1)
p. add_argument ('-- num_episodes ', type = int default = 15000)  , default = 0.99999)
                                                                 ,
p. add_argument ('-- memory_size ', type =int 30 p. add_argument ('--   , default = 100000)
burnin ', type =int                                              , default = 1000)
    p. add_argument ('-- learn_every ', type =int p.add_argument ('--    , default =1)
    sync_every ', type =int p.add_argument ('-- log_path ', type =str 34 , default = 100)
    p. add_argument ('-- save_dir ', type = str                  , default ='tensor board ')
                                                                 , default = 'trained_models ')
    p. add_argument ('-- save_interval ', type =int default = False )  , default = 70000)
p. add_argument ('--render ', type =bool                         ,
# arguments for further training and supervised data:
    p. add_argument ('-- load_path ', type =str                  ,
                        default =' trained_models / model . chkpt ')
p. add_argument ('-- continue_training ', type =bool p. add_argument ('-- new_epsilon,  default = False )
    ', type =bool default = True )                               ,
    p. add_argument ('-- create_superv_data ', type =bool 43 p. add_argument ('--    , default = False )
save_dir_superviced_data ', type =str                            ,
                        default =' supervised_data /')
    p. add_argument ('-- load_file_superviced_data ', type =str default ='benchmark_X/  ,
                        training_data_benchmark.obj ')
p. add_argument ('-- learn_from_data ', type =bool 48 p. add_argument ('--    , default = True )
supervised_steps ', type =int 49 p. add_argument ('-- print_state_2_cli ', type =bool  , default = 50000)
                                                                 , default = False )
```

```
50
51 args = parser . parse_args()
52 return args
```

Now a training routine is defined (Prog. 4.8). At the beginning due to the summary of the network is added to the passed parameters for documentation purposes (net_summary.txt) generated and output. Next, if desired, pre-training is started. Here, the network is based on already existing data trained, which either by the script 'manual mode.py' or autonomously via generated by the 'Benchmark Agent'. In the next step, the training for the specified number of episodes are carried out. How it works should be clear from the code, so no further comments will be made on this.

Program 4.8: Training Routine: Training

```
1 def train ( opt ):
2 writer = SummaryWriter(opt.log_path)
3 env = Ruetris ( opt )
4
5     if not opt . reward_action_dict :
6         state_dim = env . state_dim
7         action_dim = env . action_dim
8         agent = Controller_Agent ( state_dim , action_dim , opt ) if \
9                         opt . action_method =='controller ' else \
10                    Placement_Agent ( state_dim , action_dim , opt )
11
12        # make a nice summary of the network :
13        model_stats = summary ( agent . net , input_size =\
14                                ( opt . batch_size , state_dim ))
15        summary_str = str( model_stats )
16        with open ('net_summary .txt ', 'w') as f:
17            f.write ( summary_str )
18
19 else :
20        agent = Benchmark_Agent ( opt )
21
22 # fill memory of Placement_Agent
23    if isinstance ( agent , Placement_Agent ) and opt . learn_from_data :
24        agents . load_memory ( opt . load_file_superviced_data )
25        supervised_steps = opt . supervised_training_steps
26        for step in range ( supervised_steps ):
27            q , lose = agent . learn ()
28            if q is not None : writer . add_scalar ('SL/ Mean ', q , step )
29            if loss is not None : writer . add_scalar ('SL/ Loss ', loss , step )
```

```
            lr = agent . optimizers . param_groups [0]['lr ']
            writer . add_scalar ('SL/LR ', lr , step )
        agents . learn_from_data = False
        agents . save (0)

episodes = opt . number_episodes

### - - - - - - - - - - - - - Main loop : training the model - - - - - - - - - - - - - -###
for epoch in range ( episodes ):
        state = env . reset()
        if not isinstance ( agent , Controller_Agent ):
            state_action_dict = env . get_state_action_dict ()

        while True :
            #1. Run agent on the state
            action = agent . take_action ( state ) if \
                                isinstance ( agent , Controller_Agent ) else \
                            agents . take_action ( state_action_dict )

            #2. Agent performs action
            p_s , next_state_action_dict , next_state , reward , done = \
                                                about . step ( action , render =opt . rendering )
            if p_s is not None : state = p_s

            #3. Remember
            agents . cache( state , next_state , action , reward , done )

            #4. Learn
            q , lose = agent . learn ( epoch )

            #5. Update state
            state = next_state
            if not isinstance ( agent , Controller_Agent ):
                state_action_dict = next_state_action_dict

            #6. Check if end of game and log important data
            if done :
                writer . add_scalar ('T/ Score ', env . score , epoch )
                writer . add_scalar ('T/Tetro ', env . parts_on_board , epoch )
                writer . add_scalar ('T/ Wasted ', env . wasted_places , epoch )
                writer . add_scalar ('T/ Holes ', env . holes , epoch )
                if not isinstance ( agent , Benchmark_Agent ):
                    writer . add_scalar ('T/ Epsilon ', agent . epsilon , epoch )
                    lr = agent . optimizers . param_groups [0]['lr ']
                    writer . add_scalar ('T/ Learning_Rate ', lr , epoch )
                if q is not None : writer . add_scalar ('T/ Mean ', q , epoch )
```

```
76                  if loss is not None : writer . add_scalar ('T/L', loss , epoch )
77                  break
```

In order to be able to start the routine from the Command Line Interface (CLI), still needs a main routine/definition (Prog. 4.9). This routine will run only when the program is started. It creates the parameter list to be transferred, which is also stored as a 'train config.txt' file and starts the workout routine. The stored parameter list is then used for the test routine and for integration into the simulation program in order to make use more user-friendly.

Program 4.9: Training Routine: Main

```
1 if __name__ == '__main__ ':
2         opt = get_args ()
3         # saves arguments to config .txt file
4         with open ('train_config .txt ', 'w') as f:
5                 json . dump ( opt . __dict__ train , f , identity =2)
6         ( opt )
```

## 4.3.2 Neural Networks

In this section, the structure of the neural networks is considered. will egg If you want to change the network architecture, this must be carried out in the corresponding 'networks' module. This affects the initialization of the weighting factors, the number of layers and other neurons per layer. It will now started again with the integration of the required libraries (Prog. 4.10). For the network primarily requires the 'nn-module' (neural network). To determine the mean value and to create network copies, the corresponding functions imported.

Program 4.10: Networks: Libraries

```
1 imported torch. nn as nn
2 imported torches. mean as mean
3 import copy
```

First a network for the determination of q/state-values is created. This Network is used by both the 'Controller Agent' and the 'Placement Agent'. It is also used on the one hand for DQL and on the other hand for Double-DQL used as described in Chap. 2.6.4.7. The method used, 'kaiming normal ()', is the He initialization presented (chap. 2.6.1.4).

Program 4.11: Networks: DQN and DDQN class

```
class Policy_QN ( nn . modules ):
define __init__ ( self , i_dim , o_dim , opt ):
        super ( Policy_DDQN , s1 = self ) . __init__ ()
        opt . statescale_l1
        nodes_l1 = int( i_dim *s1 )
        s2 = opt . statescale_l2
        nodes_l2 = int( i_dim *s2 )

        self . online = nn . Sequential (\
                        nn . Linear ( i_dim nn . , nodes_l1 ) , nn . ReLU () ,
                        Linear ( nodes_l1 , nodes_l2 ) , nn . ReLU () ,
                        nn . Linear(nodes_l2 , o_dim ))

        self . _create_weights ()
        self . target = copy . deepcopy ( self . online )

        # freeze parameters of self . target
        for p in self . target . parameters ()
            p. requires_grad = False

define _create_weights ( self ):
        for m in self . module():
            if isinstance (m , nn . Linear ):
                nn . initial . kaiming_normal_ (m . weight , mode ='fan_in ',
                                                nonlinearity ='relu ')
                nn . initial . constant_ ( m. bias , 0)

def forward (self , state , model ='online '):
        if model == 'online ':
            return self . online ( state )
        elif model == 'target ':
            return self . target ( state )
```

Next, a network exclusively for the determination of q-values
created. This is the architecture for the dueling double DQL (Section 2.6.4.7), where the q-
values are determined via advantage and state value. This network is used exclusively by
for initializing the weighting factors has the same structure as in the class
'Policy QN' and has therefore been omitted.

Program 4.12: Networks: DDDQN class

```
class Policy_DDDQN ( nn . modules ):
define __init__ ( self , i_dim , o_dim , opt ):
        super ( Policy_DDDQN , self ). __init__ ()
```

```
        s1 = opt . statescale_l1
        nodes_l1 = int( i_dim *s1 )
        s2 = opt . statescale_l2
        nodes_l2 = int( i_dim *s2 )

        # define online network
        self . fc_online = nn . Sequential (\
                                    nn . Linear ( i_dim , nodes_l1 ) , nn . ReLU () ,
                                    nn . Linear(nodes_l1 , nodes_l2 ) , nn . ReLU () )
        self . value_online = nn . Sequential (\
                                        nn . Linear(nodes_l2 , 1) )
        self . adv_online = nn . Sequential (\
                                    nn . Linear(nodes_l2 , o_dim ))

        self . _create_weights ()

        # creating target network
        self . fc_target = copy . deepcopy ( self . fc_online )
        self . value_target = copy . deepcopy ( self . value_online )
        self . adv_target = copy . deepcopy ( self . adv_online )

        # freeze parameters of target network
        for p in self . fc_target . parameters ():
            p. requires_grad = False
        for p in self . value_target . parameters ():
            p. requires_grad = False
        for p in self . adv_target . parameters ():
            p. requires_grad = False

def forward (self , state , model ='online '):
        if model == 'online ':
            x = self . fc_online ( state )
            v = self . value_online (x)
            adv = self . adv_online (x)
            adv_avg = mean ( adv q = v + , dim=1 , keepdims = True )
            adv - adv_avg

        if model == 'target ':
            x = self . fc_target ( state )
            v = self . value_target (x)
            adv = self . adv_target (x)
            adv_avg = mean ( adv q = v + , dim=1 , keepdims = True )
            adv - adv_avg

        return q
```

## 4.4 AI Training

In this chapter some parameter sets are chosen to start the training routine. For reasons of space, however, only the changed values are shown
listed in the tables provided for this purpose. The complete parameter sets,
but without long file paths, are listed in Tables 12.7, 12.8, 12.9 and 12.10 in the Annex visible and correspond to the information content from the generated 'train_config.txt' files. The number of free parameters of the network is dem 'net_summary.txt' file. For more details, please refer to the Github repository [41]. For the initial parameter choice
the suggestions of a Mathworks document [45] were used. Adam (Eq. 2.16) is used as the optimizer and the values resulting from the updates of the weighting factors are displayed per episode. These values are:

- Epsilon: Parameter for the selection of random actions according to the epsilon greedy policy (Section 2.6.4.2 , Equation 2.45).
- Holes: Number of empty slots after the end of the episode.
- Learning Rate: Parameter ÿ from chap. 2.6.1.3.
- Loss: Evaluated cost functional (chap. 2.6.1.5) for a batch from the memory after completing the episode.
- Score: Accumulated rewards after completing the episode (Eq. 2.44, 3.2).
- Tetrominos: Number of components discarded after completing the episode.
- Wasted Places: Number of places that no longer allow further placement due to unfavorable placement (chap. 3.4.2.1).
- Mean: The average value of the network output (state or q value) for a ¨ batch.

### 4.4.1 Placement Agent

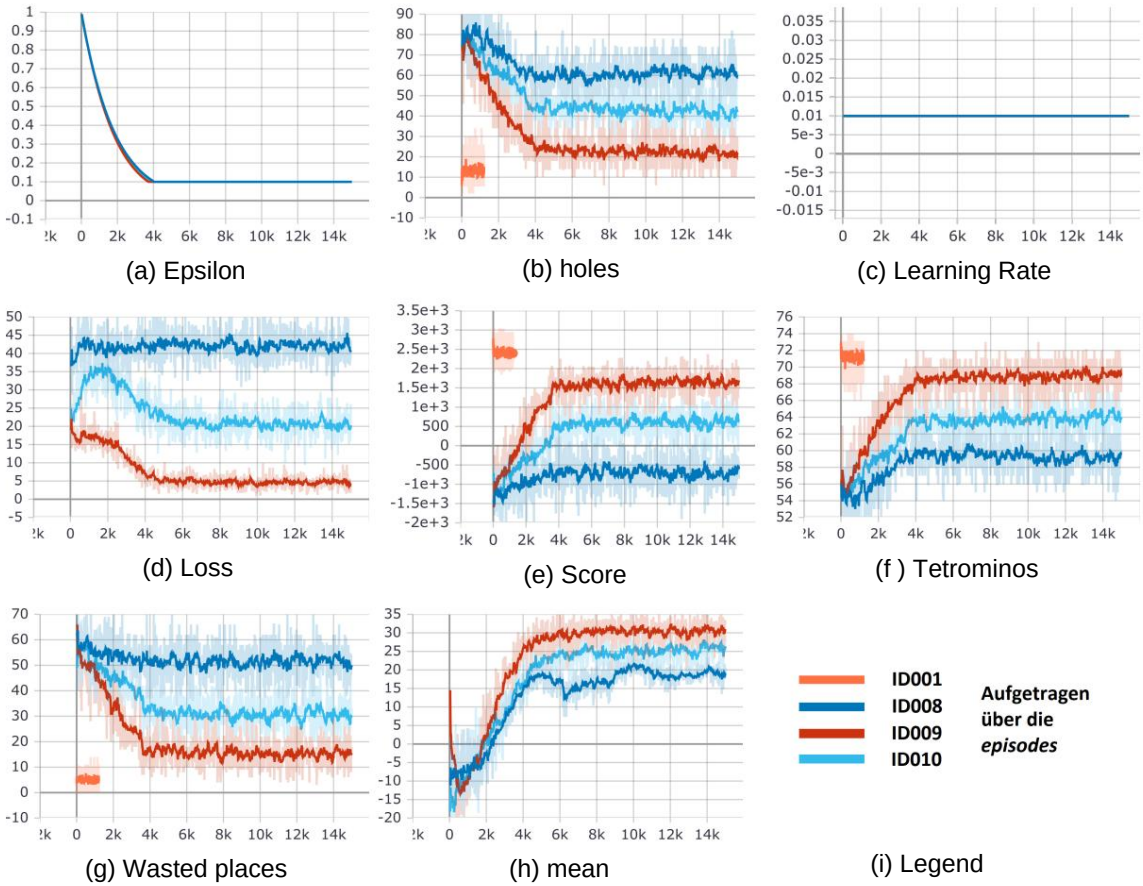
The 'Placement Agent' (Section 3.4.2.2) is used for the first test execution. The parameters used are in Table 4.1 and
4.2. The experiment with ID number 001 is the 'Benchmark Agent' which was used to generate (s,s',r,d)-tuples. Because of
Most of the parameters are also not relevant here. These tuples were later used for the pre-training phase (Prog. 4.8),

first, however, the differences and effects of the various state representations (Section 3.4.2.3) are considered. For this purpose, training parameters are chosen which differ only in the observations (Table 4.1).

Table 4.1: Parameter sets: placement agent

| Label ID | ID001 | ID008 | ID009 | ID010 |
|---|---|---|---|---|
| -state report | - | full float | reduced | full bool |
| -state scale l1 | - | 10 | 60 | 5 |
| -state scale l2 | - | 6 | 30 | 3 |
| -batch size | - | 128 | 128 | 128 |
| -lr ÿ | - | 0.01 | 0.01 | 0.01 |
| -lr decay | - | 1.0 | 1.0 | 1.0 |
| -final lr frac | - | - | - | - |
| -gamma ÿ | - | 0.2 | 0.2 | 0.2 |
| -initial epsilon | - | 1.0 | 1.0 | 1.0 |
| – final epsilon | - | 0.1 | 0.1 | 0.1 |
| -epsilon decay | - | 0.99999 | 0.99999 | 0.99999 |
| -supervised | - | false | false | false |
| number of weighting factors | - | 19 600 509 | 746 401 | 22 398 919 |

The curves resulting from these parameters, which are defined in Prog. 4.8 were shown in Fig. 4.15. These graphs show a clear favorite: the status display 'reduced'. This can be explained by the fact that the reward can be indirectly determined more easily from this representation than from the others representations, even if the information on the porosity (Chap. 3.4.2.1) is completely get lost. Furthermore, it is interesting that the representation with only boolean variables is superior to floating point representation, although the number of weighting factors is about the same. The behavior of this three agents is very different throughout and was created for documentation purposes Added to created Youtube playlist [40].




(a) Epsilon (b) holes (c) Learning Rate

(d) Loss (e) Score (f ) Tetrominos

(g) Wasted places (h) mean (i) Legend

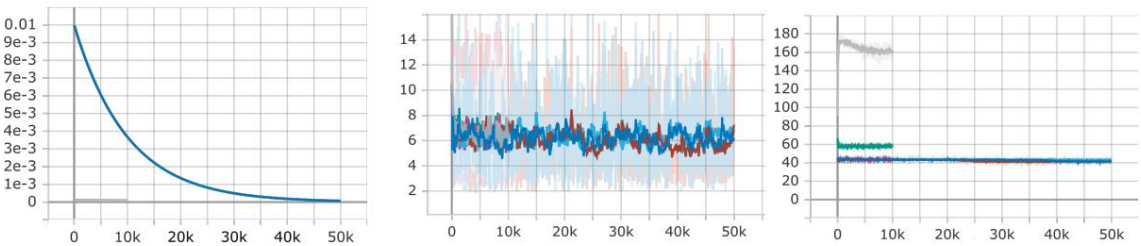
Figure 4.15: Main training: placement agent

The question naturally arises as to how the poor performance caused by the 'full float' state representation can be improved. Therefor the concept of the pre-training phase was implemented, which was used for the first tests. The parameters of these test procedures can be found in Tab. 4.2. The variance in attitudes is intended to provide further insight into learning behavior enable, such as the effect of the neuron density or the batch size.

Table 4.2: Parameter sets: Placement Agent SL

| Designation ID | ID001 | ID002 | ID003 | ID004 | ID005 | ID006 | ID007 |
|---|---|---|---|---|---|---|---|
| -state scale l1_ | - | 10 | 5 | 20 | 5 | 5 | 5 |
| -state scale l2_ | - | 6 | 3 | 12 | 3 | 3 | 3 |
| -batch size | - | 128 | 256 | 128 | 128 | 128 | 128 |
| -lr ÿ | - | 0.01 | 0.0001 | 0.01 | 0.01 | 0.01 | 0.01 |
| -lr decay | - | 0.9999 | 0.99999 | 0.9999 | 0.9999 | 0.9999 | 0.9999 |
| -final lr frac | - | 0.001 | 0.01 | 0.001 | 0.001 | 0.01 | 0.01 |
| -gamma ÿ | - | 0.2 | 0.8 | 0.2 | 0.2 | 0.2 | 0.4 |
| -initial epsilon | - | 0.01 | 0.5 | 0.01 | 0.01 | 0.5 | 0.4 |
| – final epsilon | - | 0.001 | 0.02 | 0.001 | 0.001 | 0.01 | 0.03 |
| -epsilon decay | - | 0.999999 | 0.999992 | 0.999999 | 0.999999 | 0.999995 | 0.99999 |
| -supervised | - | True | True | True | True | True | True |
| - supervised steps | - | 10000 | - | 50000 | 50000 | 10000 | 10000 |

As a result of the pre-training phase, additional graphs are now available in addition to the previous ones (Fig. 4.16), which show the learning rate, loss and the mean value of the States in the batch plotted against the gradient calls. The loss, which is decisive for the updates of the weighting factors (Section 2.6.1.3), reaches a small value within a few iterations. The neural Network, at least that is the idea, the behavior of the 'benchmark agent' should now have been shaped.




(a) Learning Rate (just (c) mean

Figure 4.16: Pre-training: Placement Agent SL

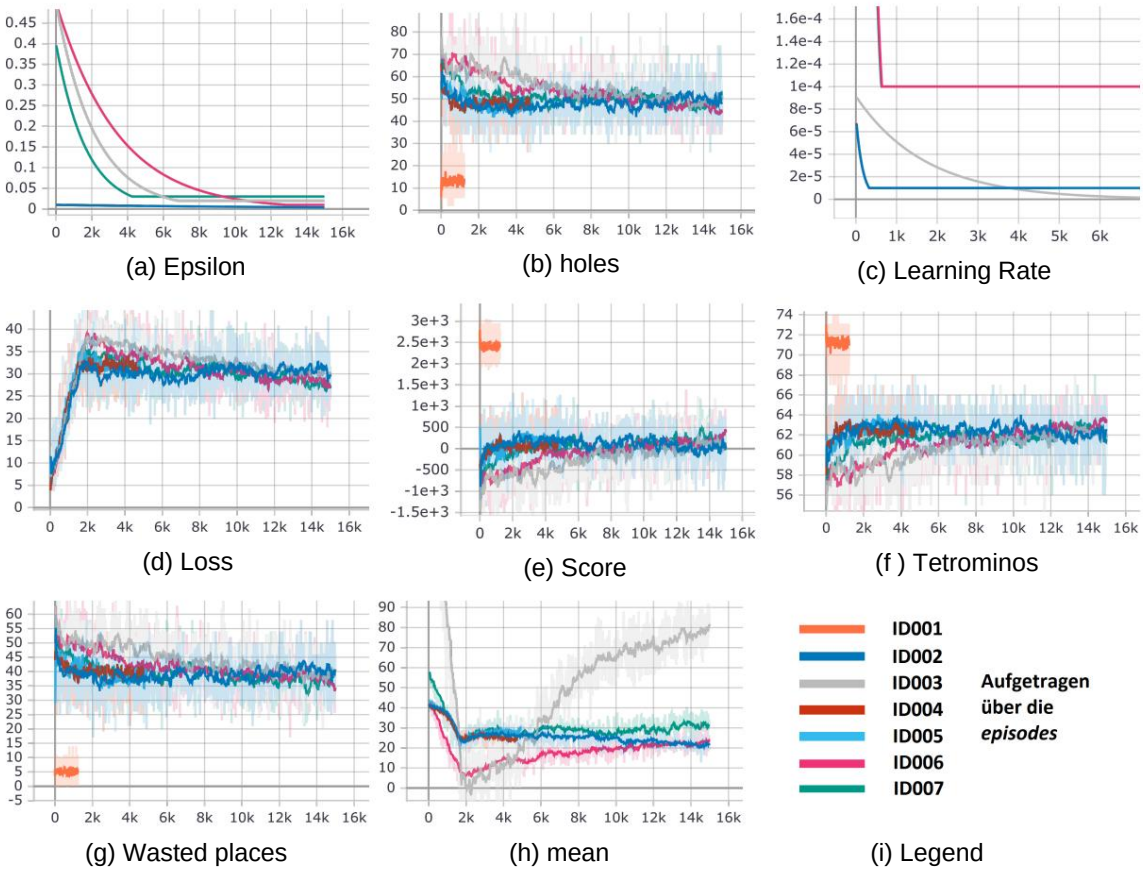



Figure 4.17: Main training: Placement Agent SL

Figure 4.17 shows the results of the main training after the completion of the pre-training. True, the expectations were that through SL the neural network is improved, but all settings of the parameters show very similar behavior (The behavior of the experiment number 002 was taken from the playlist added). The attempts with the IDs 006, 007 and 003 cannot yet be converged out and were a further test procedure require. Nevertheless, the loss was reduced from around 40 to 30 and the Score improved from around -750 to +100. For further comparisons, please refer to the corresponding graphs (Figs. 4.15 and 4.17).

### 4.4.2 Controller Agent

The 'controller agent' is used for further investigations. For the corresponding interaction method (Section 3.4.2.2) rewards for the individual actions had to be added to the environment. Table 4.3 assigns the actions defined in the environment to the corresponding rewards. The expression 'overlapping' is true if the part now hovers over an already occupied slot.

The expression 'boundary' is true if the component has left the charging area (matrix). The expression 'OK' at this point only means that neither of the two conditions is true. For further details, please refer to the Github repository [41].

Table 4.3: Rewards of the individual actions

| action | move action() | red action() | table action() | place action() | – |
|---|---|---|---|---|---|
| Reward if overlapping | -2 | -2 | - | -500 | |
| Reward if boundary | -4 | -4 | - | - | |
| Reward if OK | -1 | -1 | -3 | Ri (Eq. 3.2) | |

The training routine of the 'controller agent' can be started with the respective rewards of the individual actions and the parameters known so far. Therefor some variations of the settings (Tab. 4.4) are calculated again. as Extension to the training routine of the 'placement agent' can now also be a network type for determining the q-values (chap. 2.6.4.7) and a random start position of the component can be selected.

Table 4.4: Parameter sets: controller agent

| Designation ID | ID011 | ID012 | | ID013 | ID014 | ID015 | ID016 |
|---|---|---|---|---|---|---|---|
| –state report | reduced | reduced | full bool | full bool | full float | reduced | – |
| -net type | dd-q-net | ddd-q-net | ddd-q-net | ddd-q-net | ddd-q-net | ddd-q-net | |
| –rnd pos & rot | False | | false | false | True | True | True |
| -state scale l1 | 60 | 60 | 60 | 5 | 5 | 10 | 60 |
| -state scale l2 | 40 | 40 | 40 | 3 | 3 | 6 | 40 |
| -batch size | 128 | 128 | 128 | 64 | 64 | 64 | 64 |
| -lr decay | 1.0 | 1.0 | 1.0 | 0.999999 | 0.999999 | 1.0 | 1.0 |
| -final lr_frac | - | - | - | 0.1 | 0.1 | - | - |
| -gamma ÿ | 0.9 | 0.9 | 0.9 | 0.8 | 0.8 | 0.5 | 0.5 |
| – final epsilon | 0.01 | 0.01 | 0.01 | 0.1 | 0.08 | 0.08 | 0.08 |

As Figure 4.18 shows, no satisfactory result can be achieved with the selected settings. In this test procedure, too, the state representation 'reduced' performs best, which, however, after the end of the
Training reveals fascinating behavioral patterns, such as the perpetual rotation of the component: Smashboy (Tab. 3.1). It is also due to this pattern of behavior that the Networks with the ID 011 and 012 collect plenty of negative points. Like after
Try to expect with the 'placement agent' here is the state representation
of 'full bool' superior to that of 'full float'. At this point it should be noted that the attempt with the ID 013, which was unfortunately somewhat lost in the graphs
is, makes a good impression as far as the solution strategy of the task is concerned. One Selection of networks is in the Youtube playlist [40].

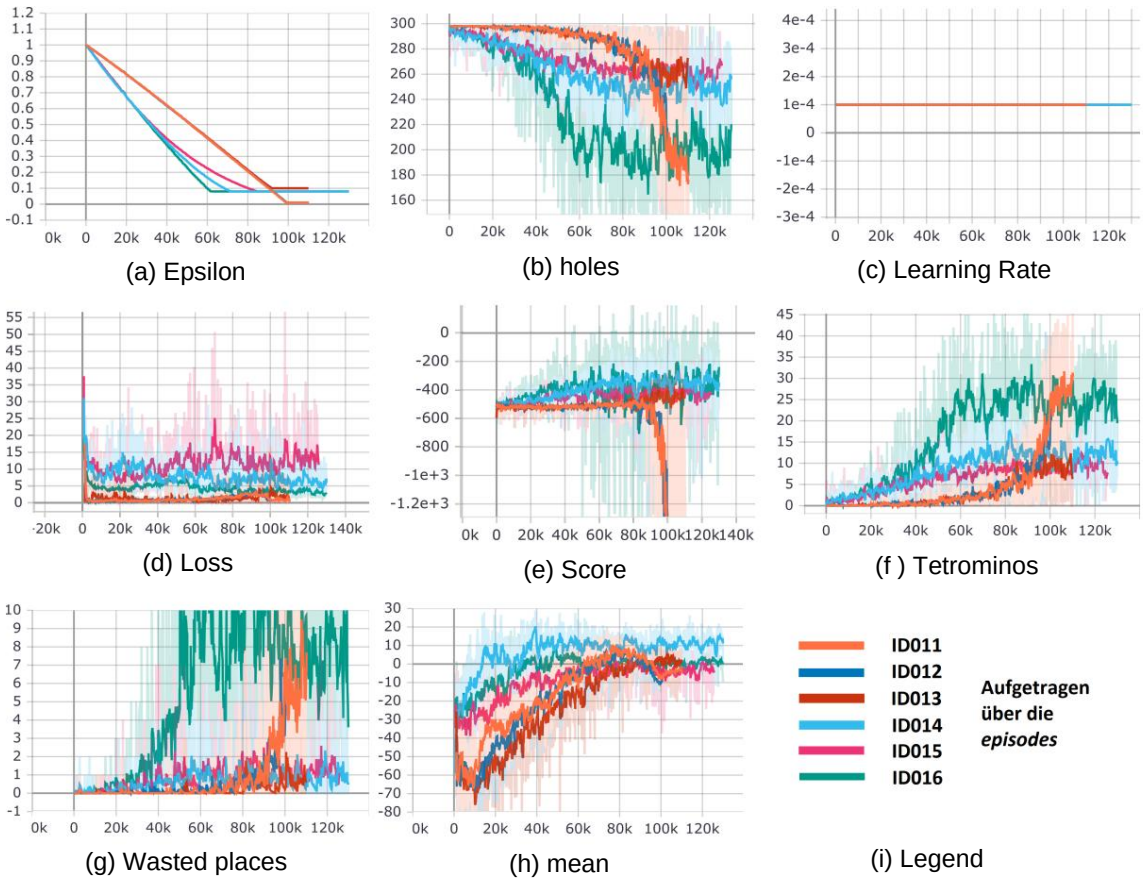



Figure 4.18: Main Training: Controller Agent

### 4.4.3 Conclusion

At the end of this chapter, the results achieved and
further possibilities for experiments were discussed. It's worth noting here that the data generated in this chapter is also on Github, however
in a separate one for reasons of storage space repository [46].

#### To the results

Even if a relatively good result was achieved in the tests with the 'Placement Agent' using the 'reduced' status display, the high score
of the 'benchmark agent' cannot be surpassed. In most cases, im
manual mode, however, this can be achieved with certain tactics (chap. 12.6). in the
In the course of carrying out the test, especially by the 'controller agent', the question arose whether it was possible to identify the component by specifying the area centroids for the network, or whether further information about this was possible
State representation had to be added. For example, one could not
only the distances of the surface normals, but also the diagonal distances of the edges use. Whether this would improve behavior could not be tested in the course of this work and is therefore considered questionable. In any case, the results obtained offer a basis for further research.

#### Other existing options

Although the possibility of loading networks that have already been saved and carrying out another training routine was implemented, this was not done in this work. Furthermore, there is the possibility of using the presented SL implementation for other state representations as well. To do this, the 'Benchmark Agent' only had to be left running while with the option provided in order to generate the necessary data. It would also be conceivable
create such an agent for the 'controller' method by specifying the storage position using the 'placement' method and the actions defined in
mouth this state, are calculated. Another, already implemented option for generating this data would be to use the script 'manual mode.py'. This script allows a human to interact with the environment and to record the actions and states.

**Further possibilities to be implemented**

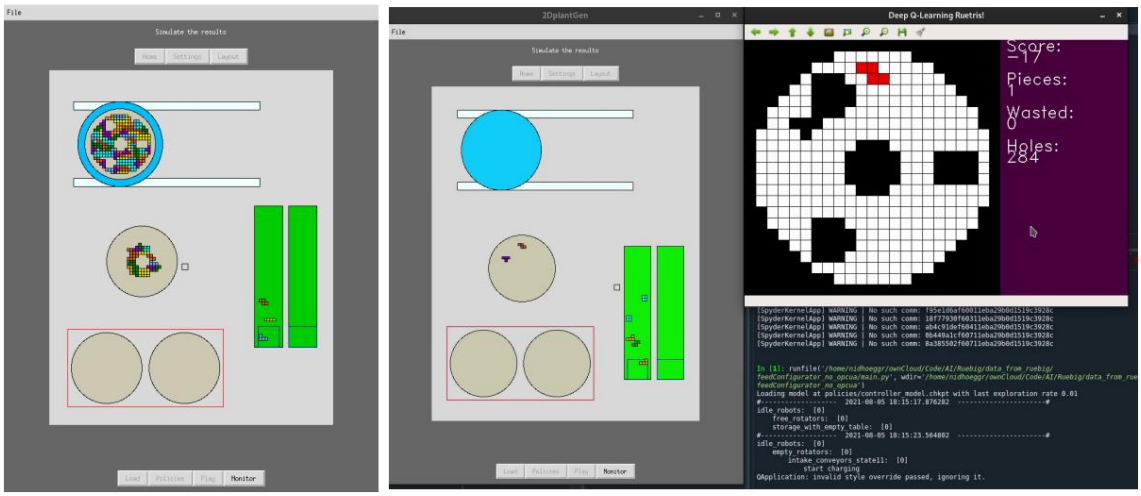
In chapter 2.6.1.8 the Raytune library was mentioned to convert the parameters of the
to optimize the network. The implementation was omitted to the effect that
since a Graphics Processing Unit (GPU) cluster would be needed to run the optimization efficiently. Such
a cluster was not available, Google Cloud had a powerful VM
however one could obtain through services like that.
Another point for possible implementations would be the extension
through different agents and network structures. For this purpose, in Chap. 2.6.4.5 several possibilities
are presented (Fig. 2.41).

**Personal opinion**

In this section I would like to add something of my own opinion.
First of all, I would like to emphasize that all the topics of this work were new territory for me, but especially
the development of the chapter ML and the following one
Conception of an implementation challenging. The complexity and the
Unfortunately, due to time resources, I wasn't able to try out all the ideas. Maybe I'll be able to refine
these chapters further in the future, won't I
another interested party finds time to deal with improvements.
At this point, I would like to mention one more thing about the strengths and weaknesses of the RL
interesting blog post [47].

# 4.5 Integration into the simulation program

In order to keep the integration into the simulation and control program simple, the 'train config.txt' file is
used for the implementation. meanwhile
With this help, switching from neural networks is reduced to renaming
of these two files and storing them in the policies folder provided for this purpose
(Fig. 3.6). In the program routines, the environment object is created accordingly during execution and
the appropriate agent is selected. In the 'Placement Agent' class, epsilon is initialized with zero and in
the 'Controller Agent'
with non-zero. The non-zero initialization is derived from the parameters described in Chap. 4.4.2
chosen for the reasons mentioned. Figure 4.19 shows how agents are used
within the simulation environment, a recording of the 'placement agent' is in playlist [40]. When using the
'Controller Agent'
the environment is called, which is controlled by the agent until the current component has been placed.
Then the movements in the simulation
accepted.




(a) Placement Agent (b) Controller Agent

Figure 4.19: Integration of the agents

# 5 Summary and Outlook

The aim of this work was to create a user-friendly interaction scheme for the charging of hardening plants and the data models required for this to find the optimal charge yourself. In order to solve this task related to the interaction scheme, a process configuration and a control logic for its control were developed. This solution was in a next step applied to a robot training cell to prove its functionality.

For the question of suitable data models for the optimal charging, the components were approximated by simple geometries. These approximated components were then determined by given positions or via detours by evaluating neural networks, stored on the charging racks. For the Cataloging of more complex components was thereby taking the form of tetrominoes used.

Necessary program routines for training the neural networks were created and used in a first iteration, which are now also available online to stand.

The novelty of this work is the application of neural networks to the tiling problem of the charging racks of hardening plants is automated and to be solved in a reasonable amount of time. A usable data model for this task could already be created in the first test execution, which, however, offers a lot of room for further improvements. This model was integrated into the configuration program that was created and could therefore be transferred to real systems.

Since the program routines created still offer some potential for improvement, would be, in a further iteration, a more comprehensive implementation within of a project team makes sense. Although the initial implementation is not designed for end use, it is suitable as a demonstrator for further projects derive from it.

### Process Flow Optimization

When creating the software, care was taken to ensure that the overall system can be mapped using states and their transitions. So would also be paved the way for an AI to learn the process flow. The idea would be to create a plant layout and to be able to export it as a learning environment for the AI. Once the learning process has been carried out, the generated policy could be returned load into the environment and assess. By doing this, a Full automation of any system layout using AI is definitely feasible.

### grappling tactics

The topic of autonomous gripping was completely ignored in this work. In order to be able to carry out the process as presented in this work, a appropriate strategy is required in this regard. A relatively simple possibility would be to equip a gripper with adaptable suction cups that conform to the shape of Adjust tetrominoes.

### Identification of further component classes

In this work, the UL concept was introduced in Chap. 2.6.3 introduced briefly, but not further used. However, this method might be suitable to identify other component classes besides cylindrical and tetromino-shaped components.
Furthermore, it would also be possible that pentominoes, hexominoes and others Polyominos [34] [48] can be used for component classification. Also would be it is conceivable to use other basic geometries for the type of classification, the platonic tiling provides us with the appropriate partial geometries for this.

# 6 List of acronyms

Adam Adaptive Momentum Estimation
BGD Batch Gradient Descent
CLI Command line interface
CNN Convolutional Neural Network
CPU Central Processing Unit
DQL Deep-Q Learning
DQN Deep-Q Network
FNN Feedforward Neural Network
GPU Graphics Processing Unit
HTTP Hypertext Transfer Protocol
HTTPS Hypertext Transfer Protocol Secure
ID identifier
IEC International Electrotechnical Commission
KI Artificial intelligence
MFC Mass Flow Controller
MGD Minibatch Gradient Descent
ML Machine Learning
MSE Mean Square Error
OPC UA Open Platform Communication Unified Architecture
POU program organization unit
RL Reinforcement Learning
RNN Recurrent Neural Network
SGD Stochastic gradient descent
SL Supervised Learning
SPS Programmable logic controller
TCP Transmission Control Protocol
TDL Temporal Difference Learning
UI UI
UL Unsupervised Learning
UML Unified Markup Language
URI Uniform Resource Identifier
URL Uniform Resource Locator
VM Virtual machine

# 7 List of Figures

# 8 List of Tables

# 9 List of Formulas

# 10 program directory

# 11 Bibliography

[1] Vajna, S'andor, Weber, Christian, Zeman, Klaus, Hehenberger, Peter, Gerhard, Detlef, and Wartzack, Sandro. CAx for Engineers: A Practical Introduction (English Edition) . Springer Vieweg, 2018.

[2] Heinrich, Berthold, Linke, Petra, and Gl¨ckler, Michael. Basics Auto tization. Springer Verlag GmbH, 2020.

[3] John. IEC 61131-3: Programming industrial automation systems: concepts and programming languages, requirements for programming systems, decision making aids. Berlin New York: Springer, 2010.

[4] Plenk, Valentin. Applied network technology compact File formats, transmission protocols and their use in Java applications. Wiesbaden: Springer Vieweg, 2017.

[5] Wikipedia of the OPCFoundation Group [http://wiki.opcfoundation.org]. Called up on April 1st, 2021.

[6] Fabian Spitzer, MSc. Framework for the configuration of intralogistics systems for the automated generation of the communication logic. 2019. Master's thesis. FH catfish.

[7] Damm, Gappmeier, Zugfil, Pl¨ob, Fiat, and St¨ortkuhl. Security Analysis OPC u.a. Available from: https://www.bsi.bund.de/SharedDocs/Downloads/DE/BSI/Publications/Studies/OPCUA/OPCUA.pdf. Technical report. federal office for security in information technology. Accessed on 06/24/2021.

[8] Randen, Hendrik Jan van, Bercker, Christian, and Fieml, Julian. Introduction in UML. Gabler, Betriebswirt.-Vlg, 2016.

[9] Alsina, Claudia. Pearls of mathematics: 20 geometric figures as starting points for mathematical explorations. Berlin: Springer Spectrum, 2015

[10] Strick, Heinz Klaus. Mathematics is beautiful. Springer Verlag GmbH, 2021

[11] Wikipedia. Kepler's conjecture [https://de.wikipedia.org/wiki/Keplersche Supposition].

[12] The Kepler Conjecture. Springer Verlag GmbH, 2011.

[13] Michelucci, Umberto. Applied Deep Learning. APRESS LP, 2018.

[14] Sanghi, Nimish. Deep Reinforcement Learning with Python. Apres, 2021.

[15] G'eron, Aur'elien. Hands-on Machine Learning with Scikit-Learn, Keras, and TensorFlow. O'Reilly UK Ltd, 2019.

[16] PyTorch. PyTorch Documentation [https://pytorch.org/docs/stable/index.html]. Accessed on 06/24/2021.

[17] Activation functions nn module [https://pytorch.org/docs/stable/nn.html]. Accessed on 07/29/2021.

[18] Ruder, Sebastian. An overview of gradient descent optimization algorithms. 2016 (). Available from arXiv: 1609.04747 [cs.LG].

[19] Glorot, Xavier. Understanding the difficulty of deep forward neural training networks. In: AISTATS. 2010

[20] Kumar, Siddharth Krishna. On weight initialization in deep neural networks. 2017 (). Available from arXiv: 1704.08863 [cs.LG].

[21] Boulila, Wadii, Driss, Maha, Al-Sarem, Mohamed, Saeed, Faisal, and Krichen, moez Weight Initialization Techniques for Deep Learning Algorithms in Remote Sensing: Recent Trends and Future Perspectives. 2021 (). Available from arXiv: 2102.07004 [cs.LG].

[22] DiPietro, Rob. A Friendly Introduction to Cross-Entropy Loss [https://rdipietro.github.io/friendly - intro - to - cross - entropy - loss/]. 2016. Accessed on 04/01/2021.

[23] Cavazza, Jacopo, and Murino, Vittorio. Active Regression with Adaptive Huber-like Loss. 2016 (). Available from arXiv: 1606.01568 [cs.LG].

[24] Hyperparameter tuning tutorial [https://pytorch.org/tutorials/beginner/hyperparameter_tuning_tutorial.html]. Accessed 07/19/2021.

[25] Gao, Tianxiang, and Jojic, Vladimir. Degrees of Freedom in Deep Neural network. 2016 (). Available from arXiv: 1603.09260 [cs.LG].

[26] Pattanayak, Santanu. Pro Deep Learning with TensorFlow. Apres, 2017.

[27] Torchvision models [https://pytorch.org/vision/stable/models.html]. upset fen on 08/15/2021.

[28] rpatrik96. Advantage Actor Critic [https://github. com/rpatrik96/pytorch-a2c/blob/master/src/model.py]. 2019. Accessed 07/19/2021.

[29] Ohrn, Anders. Image Clustering Implementation with PyTorch [ https://towardsdatascience.com/image-clustering-implementation-with-pytorch 587af1d14123]. Accessed 07/31/2021.

[30] OpenAI. Kind of RL Algorithms [https://spinningup.openai.com/en/latest/ spinningup/rlintro2.html]. Accessed on 06/24/2021.

[31] Shaul, Tom, Quan, John, Antonoglou, Ioannis, and Silver, David. Prioritized Experience Replay. 2015 (). Available from arXiv: 1511.05952 [cs.LG].

[32] Hasselt, Hado van, Guez, Arthur, and Silver, David. Deep Reinforcement Learning with Double Q-learning. 2015 (). Available from arXiv: 1509.06461 [cs.LG].

[33] Wang, Ziyu, Schaul, Tom, Hessel, Matteo, Hasselt, Hado van, Lanctot, Marc, and Freitas, Nando de. Dueling Network Architectures for Deep Reinforcement Learning. 2015 (). Available from arXiv: 1511.06581 [cs.LG].

[34] Golomb, Solomon. Polyominoes: puzzles, patterns, problems, and packings. Princeton, NJ: Princeton University Press, 1994.

[35] When a claim was made that each Tetris block has a name [ http://www.8-bitcentral.com/ blog/2019/tetrisBlockNames.html]. Accessed 08/14/2021.

[36] Siemens. SIMATIC Robot Integration for KUKA [https:// support.industry. siemens.com/ cs/document/109482123]. Accessed on 06/23/2021.

[37] Siemens. OPC UA methods for the SIMATIC S7-1500 [https://support.indust ry.siemens.com/cs/document/109756885]. Accessed on 06/23/2021.

[38] Siemens. Function manual SIMATIC S7 1500 communication. 2019

[39] Library, Free OPC UA. GitHub repository of asyncua [https://github.com/FreeOpcUa/ opcua-asyncio ]. Accessed 8/3/2021.

[40] Brandstaetter, Christian. Videos of this work [https://www. youtube. com/ channel/ UChI56p6C45m HcJ eteWmgA/playlists?view=1&sort=dd&shelf id= 0]. Accessed on 08/02/2021.

[41] Brandstaetter, Christian. GitHub repository of this work [https://github.com/codecalypse/ ruetris/ ]. Accessed on 07/30/2021.

[42] Reinforcement learning (DQN) tutorial [https://pytorch.org/tutorials/intermediate/ reinforcement q learning.html]. Accessed on 06/23/2021.

[43] Train a mario playing RL agent (DDQN) [https : / / pytorch . org/tutorials/ intermediate/ mariorltutorial.html]. Accessed on 06/23/2021.

[44] Uvipen. Deep Q-learning for playing Tetris [https://github.com/uvipen/Tetris deep-Q-learning-pytorch]. Accessed on 06/23/2021.

[45] MathWorks. Reinforcement Learning Toolbox User's Guide [https://www.qiu cle.cn/wp-content/uploads/2020/05/reinforcement user guide.pdf]. invoked on 07/30/2021.

[46] Brandstaetter, Christian. GitHub repository of results [https://github.com/ codecalypse/ruetris results]. Accessed on 08/06/2021.

[47] Irpan, Alex. Deep Reinforcement Learning Doesn't Work Yet [https://www. alexirpan.com/2018/02/14/rl-hard.html]. 2018

[48] Wikipedia. Polyomino [https://de.wikipedia.org/wiki/Polyomino].

# 12 Appendix

## 12.1 Fundamentals of Statistics

Definition Expected value of a discrete random variable X.

$$E[X] = X_{k\ddot{y}Dx} \; kP(X = k) \tag{12.1}$$

Linearity property of the expectation.

$$E[\ddot{y}X + \ddot{y}Y ] = \ddot{y}E[X] + \ddot{y}E[Y ] \tag{12.2}$$

## 12.2 Hardware and software used

The configuration of the development environment can be found in the following tables.
The configuration of the AI development environment is located as. 'env.yml'-
File on Github [41].

Table 12.1: UI development environment setup

| Framework/ Library/ OS version | |
|---|---|
| scipy | 1.6.1 |
| Tkinter | - |
| numpy | 1.19.3 |
| windows 10 | 10.0.18363 |
| Spyder | 4.2.3 |
| python | 3.7.9 |
| Async OPCUA | 09/09/90 |

Table 12.2: AI development environment setup

| Framework/ Library/ OS/ IDE/ Hardware | version |
|---|---|
| PyTorch | 1.9.0 |
| Conda | 4.9.2 |
| python | 3.6.12 |
| TensorboardX | 2.4.0 |
| Linux Manjaro Gnome | 21.1.0 |
| Linux kernel | 5.10.53-1 |
| numpy | 1.19.5 |
| OpenCV | 4.1.0 |
| Spyder | 4.2.1 |
| motherboard | ASUS Prime |
| CPU (overclocked) | Intel Core i7-7800X 4.27GHz |
| GPU (overclocked) | Geforce RTX 2080 2.1GHz |
| GPU (first died) | Geforce RTX 3070TI |
| RAM (overclocked) | Corsair 32GB 3.2GHz |

## 12.3 States of the Simulation Member Classes

In the following the states are given, binary or decimal coded, which the respective class objects from the 'Simulation members' module to.

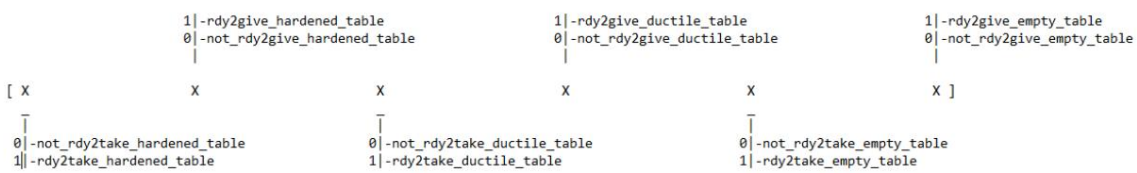



Figure 12.1: States of the class 'Plant'

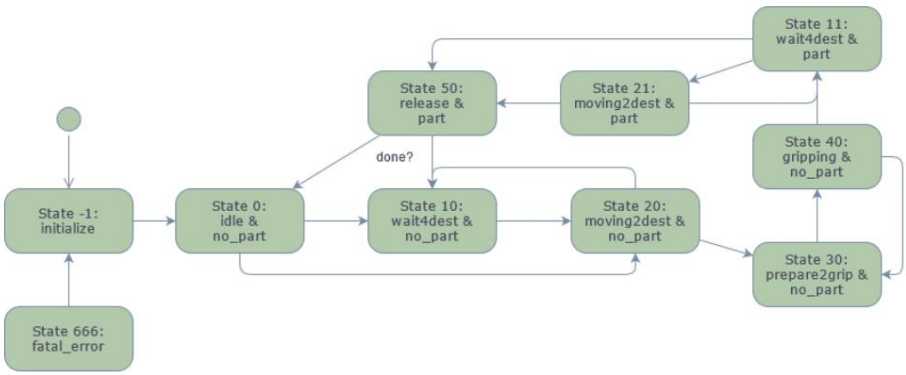



Figure 12.2: State diagram of the 'Robot' class

```
                1|-part
                0|-no_part
                 |
[ X               X ]
  _
  |
 0|-idle
 1|-wait4dest
 2|-moving2dest
 3|-prepare2grip
 4|-gripping
 5|-release
```

Figure 12.3: States of the 'Robot' class

```
                                    2|-backwards
                                    1|-forward
                                    0|-off
                                     |
[ X                               X ]
  _
  |
 0|-not_rdy2give_part
 1|-rdy2give_part
```

Figure 12.4: States of the 'Conveyor' class

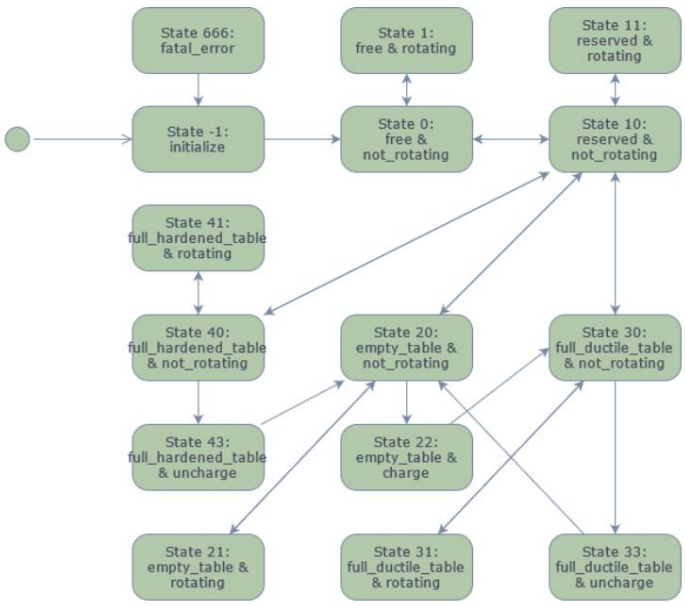



Figure 12.5: State diagram of the 'Rotator' class

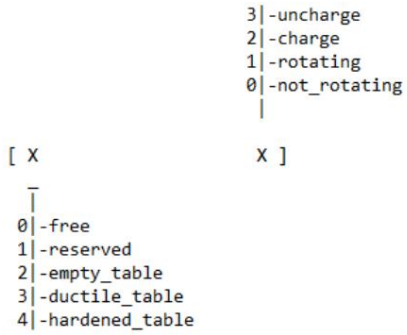



Figure 12.6: States of the 'Rotator' class

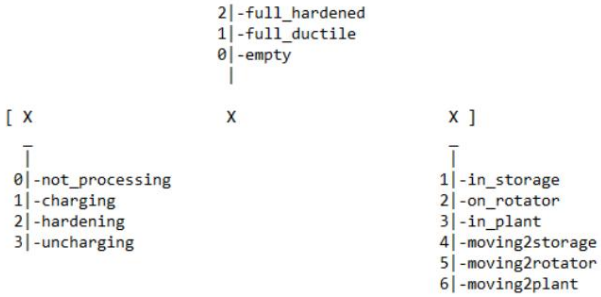



Figure 12.7: States of the 'Table' class

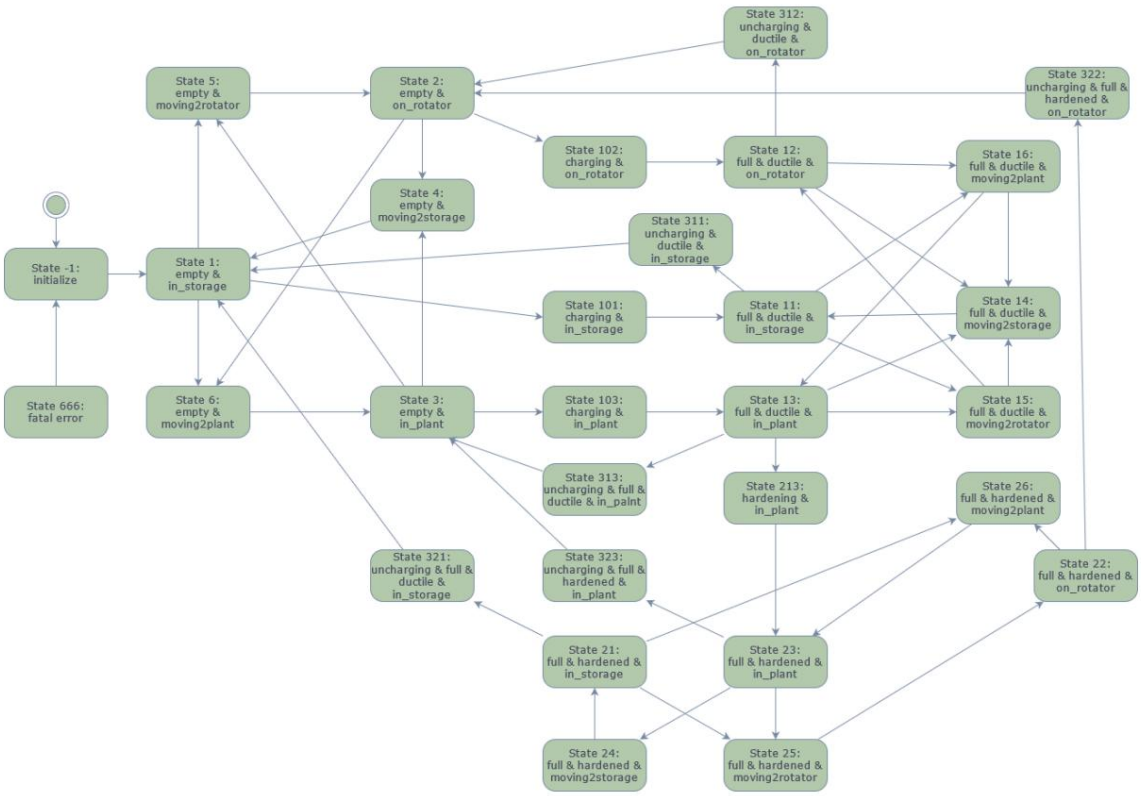



Figure 12.8: State diagram of the 'Table' class

## 12.4 OPC UA data model

The following tables show the structure of the communication to the PLC of the training cell. The designation MFC corresponds to the created simulation program.

Table 12.3: Struct: MFC to_Robot

| nodes | data type | description |
|---|---|---|
| Job_cmd | structure | Command Structure for Pick&Place Application |
| LifeBit | Boolean | square-wave signal as a sign of life (min. 0.2 Hz) |
| NewDataConfirmed | Boolean | Acknowledgment bit when new data is received |
| NewDataValid | Boolean | Information bit about sending new data |

Table 12.4: Struct: Robot to_MFC

| nodes | data type | description |
|---|---|---|
| Job_state | structure | Information structure of job execution |
| LifeBit | Boolean | square-wave signal as a sign of life with 0.5 Hz |
| NewDataConfirmed | Boolean | Confirmation bit when new data is received |
| NewDataValid | Boolean | Information bit about sending new data |
| Ready | boolean | Ready for a job execution |

Table 12.5: Struct: Job cmd

| nodes | data type | description |
|---|---|---|
| Target position struct | | Target or storage position |
| Source Position Structure | | Start or pick-up position |
| Target ID | integer | Reference system (ID) of the storage position |
| SourceID | integer | Reference system (ID) of the pick-up position |
| JobType | Integer task definition: 1)Pick, 2)Place, 3)Pick&Place | |
| execute | boolean | Execution command bit |

Table 12.6: Struct: Job state

| nodes | data type | description |
|---|---|---|
| busy | boolean | Logical 1 when busy |
| done | boolean | Logical 1 when job is processed |
| errors | boolean | Logical 1 if errors occurred |
| Gripper Occupied Boolean | | Logical 1 if the component is in the gripper |
| Job Plausible Boolean | | Logical 1 if the transferred data makes sense |
| pick busy | boolean | Logical 1 for gripping routine |
| pick done | boolean | Logic 1 when gripping routine is finished |
| pick error | Boolean | Logical 1 if there is an error during the grab routine |
| Place Busy | boolean | Logical 1, with storage routine |
| place done | boolean | Logical 1 when the storage routine is complete |
| PlaceError | Boolean | Logical 1 on errors during the filing routine |

## 12.5 Parameter lists of the test execution

The parameter settings for the test implementation follow. Further details can be found in the Github repository [46].

Table 12.7: Parameter settings A

| Label ID | ID001 | ID002 | ID003 | ID004 |
|---|---|---|---|---|
| - table dia | 420 | 420 | 420 | 420 |
| - block size | 20 | 20 | 20 | 20 |
| -state report | 'full float' 'full float' 'full float' 'full float' | | | _ |
| – action method | - | 'placement' 'placement' 'placement' | | |
| -rnd pos | - | false | false | false |
| –rnd red | - | false | false | false |
| – reward action dict | True | false | false | false |
| -c1 | 2 | 2 | 2 | 2 |
| -c2 | 50 | 50 | 50 | 50 |
| -c3 | 3 | 3 | 3 | 3 |
| -c4 | 5 | 5 | 5 | 5 |
| -net type | - | 'dq-net' | 'dq-net' | dq-net |
| -state scale l1 | - | 10 | 10 | 20 |
| -state scale l2 | - | 6 | 6 | 12 |
| -batch size | - | 128 | 128 | 128 |
| -lr ÿ | - | 0.01 | 0.01 | 0.01 |
| -lr decay | - | 0.9999 | 1.0 | 0.9999 |
| -final lr frac | - | 0.001 | - | 0.001 |
| -gamma ÿ | - | 0.2 | 0.2 | 0.2 |
| -initial epsilon | - | 0.01 | 1 | 0.01 |
| – final epsilon | - | 0.001 | 0.1 | 0.001 |
| -epsilon decay | - | 0.999999 | 0.99999 | 0.999999 |
| -num episodes | 1.3e3 | 15e3 | 15e3 | 15e3 |
| -memory size | 1e5 | 1e5 | 1e5 | 1e5 |

| | | | | |
|---|---|---|---|---|
| -burnin | - | 1000 | 1000 | 1000 |
| –learn every | - | 1 | 1 | 1 |
| -sync every | - | 100 | 100 | 100 |
| – save_interval | 100 | 7e4 | 7e4 | 7e4 |
| -render | True | false | false | false |
| - load path | - | - | - | - |
| -continue training | - | false | false | false |
| -new epsilon | - | - | - | - |
| –create supervised data | True | false | false | false |
| –learn from benchmark | - | True | false | True |
| - supervised training steps | - | 5e4 | - | 5e4 |
| – print state 2 cli | false | false | false | false |

Table 12.8: Parameter settings B

| Label ID | ID005 | ID006 | ID007 | ID008 |
|---|---|---|---|---|
| - table dia | 420 | 420 | 420 | 420 |
| - block size | 20 | 20 | 20 | 20 |
| -state report | 'full float' | 'full float' | 'full float' | 'full float' |
| – action method | 'placement' | 'placement' | 'placement' | 'placement' |
| -rnd pos | false | false | false | false |
| –rnd red | false | false | false | false |
| – reward action dict | false | false | false | false |
| -c1 | 2 | 2 | 2 | 2 |
| -c2 | 50 | 50 | 50 | 50 |
| -c3 | 3 | 3 | 3 | 3 |
| -c4 | 5 | 5 | 5 | 5 |

| -net type | 'dq-net' | 'dq-net' | 'dq-net' | dq-net |
|---|---|---|---|---|
| -state scale l1 | 5 | 5 | 5 | 10 |
| -state scale l2 | 3 | 3 | 3 | 6 |
| -batch size | 128 | 128 | 128 | 128 |
| -lr ÿ | 0.01 | 0.01 | 0.01 | 0.01 |
| -lr decay | 0.9999 | 0.9999 | 0.9999 | 1.0 |
| -final lr frac | 0.001 | 0.01 | 0.01 | - |
| -gamma ÿ | 0.2 | 0.2 | 0.4 | 0.2 |
| -initial epsilon | 0.01 | 0.5 | 0.4 | 0.01 |
| – final epsilon | 0.001 | 0.01 | 0.03 | 1.0 |
| -epsilon decay | 0.999999 0.999995 | | 0.99999 | 0.99999 |
| -num episodes | 15e3 | 15e3 | 15e3 | 15e3 |
| -memory size | 1e5 | 1e5 | 1e5 | 1e5 |
| -burnin | 1000 | 1000 | 1000 | 1000 |
| –learn every | 1 | 1 | 1 | 1 |
| -sync every | 100 | 100 | 100 | 100 |
| – save interval | 7e4 | 7e4 | 7e4 | 7e4 |
| -render | false | false | false | false |
| - load path | - | - | - | - |
| -continue training | false | false | false | false |
| -new epsilon | - | - | - | - |
| –create supervised data | false | false | false | false |
| –learn from benchmark | True | True | True | false |
| – supervised training steps | 5e4 | 1e4 | 1e4 | - |
| – print state 2 cli – | false | false | false | false |

Table 12.9: Parameter settings C

| Label ID | ID009 | ID010 | ID011 | ID012 |
|---|---|---|---|---|
| - table dia | 420 | 420 | 420 | 420 |
| - block size | 20 | 20 | 20 | 20 |
| -state report | 'reduced' 'full bool' 'reduced' 'reduced' | | | |
| – action method | 'placement' 'placement' 'controller' 'controller' | | | |
| -rnd pos | false | false | false | false |
| –rnd red | false | false | false | false |
| – reward action dict | false | false | false | false |
| -c1 | 2 | 2 | 2 | 2 |
| -c2 | 50 | 50 | 50 | 50 |
| -c3 | 3 | 3 | 3 | 3 |
| -c4 | 5 | 5 | 5 | 5 |
| -net type | 'dq-net' | 'dq-net' 'dd-q-net' ddd-q-net | | |
| -state scale l1 | 60 | 5 | 60 | 60 |
| -state scale l2 | 30 | 3 | 40 | 40 |
| -batch size | 128 | 128 | 128 | 128 |
| -lr ÿ | 0.01 | 0.01 | 1e-4 | 1e-4 |
| -lr decay | 1.0 | 1.0 | 1.0 | 1.0 |
| -final lr frac | - | - | - | - |
| -gamma ÿ | 0.2 | 0.2 | 0.9 | 0.9 |
| -initial epsilon | 1.0 | 1.0 | 1.0 | 1.0 |
| – final epsilon | 0.1 | 0.1 | 0.01 | 0.01 |
| -epsilon decay | 0.99999 | 0.99999 | 0.999999 0.999999 | |
| -num episodes | 15e3 | 15e3 | 4e5 | 1.1e5 |
| -memory size | 1e5 | 1e5 | 1e5 | 1e5 |

| | | | | |
|---|---|---|---|---|
| -burnin | 1000 | 1000 | 1e4 | 1e4 |
| –learn every | 1 | 1 | 3 | 3 |
| -sync every | 100 | 100 | 1e4 | 1e4 |
| – save_interval | 7e4 | 7e4 | 7e5 | 1e6 |
| -render | false | false | false | false |
| - load path | - | - | - | - |
| -continue training | false | false | false | false |
| -new epsilon | - | - | - | - |
| –create supervised data | false | false | false | false |
| –learn from benchmark | false | false | false | false |
| - supervised training steps | - | - | - | - |
| – print state 2 cli_ | false | false | false | false |

Table 12.10: Parameter settings D

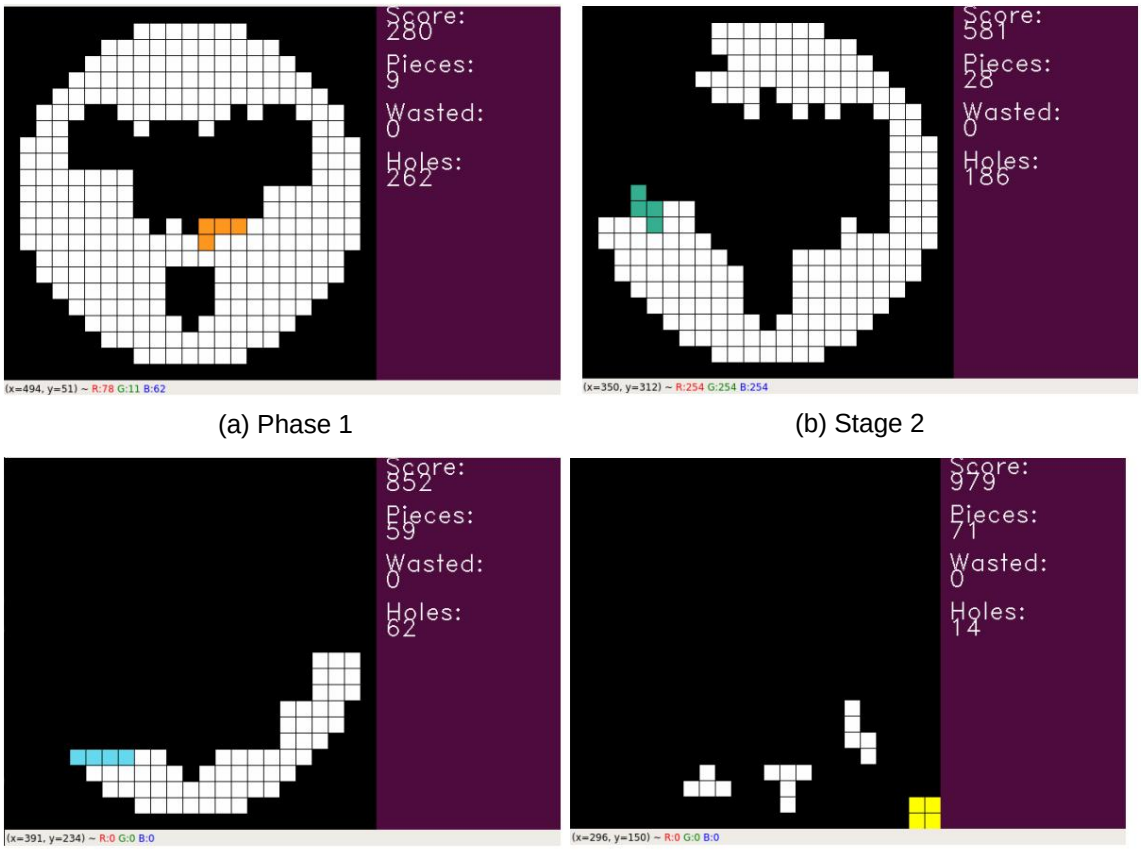
| Label ID | ID013 | ID014 | ID015 | ID016 |
|---|---|---|---|---|
| - table dia | 420 | 420 | 420 | 420 |
| - block size | 20 | 20 | 20 | 20 |
| -state report | 'full bool' | 'full bool' | 'full float' | 'reduced' |
| – action method | 'controller' | 'controller' | 'controller' | 'controller' |
| -rnd pos | false | True | True | True |
| –rnd red | false | True | True | True |
| – reward action dict | false | false | false | false |
| -c1 | 2 | 2 | 2 | 3 |
| -c2 | 50 | 50 | 50 | 50 |
| -c3 | 3 | 3 | 3 | 3 |
| -c4 | 5 | 5 | 5 | 5 |

| | | | | |
|---|---|---|---|---|
| -net type | 'ddd-q-net' | 'ddd-q-net' | 'ddd-q-net' | ddd-q-net |
| -state scale l1 | 5 | 5 | 10 | 60 |
| -state scale l2 | 3 | 3 | 6 | 40 |
| -batch size | 64 | 64 | 64 | 64 |
| -lr ÿ | 1e-4 | 1e-4 | 1e-4 | 1e-4 |
| -lr decay | 0.999999 | 0.999999 | 1.0 | 1.0 |
| -final lr frac | 0.1 | 0.1 | - | - |
| -gamma ÿ | 0.8 | 0.8 | 0.5 | 0.5 |
| -initial epsilon | 1.0 | 1.0 | 1.0 | 1.0 |
| – final epsilon | 0.1 | 0.08 | 0.08 | 0.08 |
| -epsilon decay | 0.999999 | 0.999999 | 0.999999 | 0.999999 |
| -num episodes | 1.1e5 | 1.3e5 | 1.3e5 | 1.3e5 |
| -memory size | 1e5 | 1e5 | 1e5 | 1e5 |
| -burnin | 1e4 | 1e4 | 1e4 | 1e4 |
| –learn every | 3 | 3 | 3 | 3 |
| -sync every | 1e4 | 1e4 | 1e4 | 1e4 |
| – save interval | 1e6 | 1e6 | 1e6 | 1e6 |
| -render | false | false | false | false |
| - load path | - | - | - | - |
| -continue training | false | false | false | false |
| -new epsilon | - | - | - | - |
| –create supervised data | false | false | false | false |
| –learn from benchmark | false | false | false | false |
| - supervised training steps | - | - | - | - |
| – print state 2 cli – | false | false | false | false |

## 12.6 Manual Interaction Strategy

Since manual interaction with the environment was implemented, own strategies for the optimal solution can also be generated. A promising strategy is briefly presented here (Fig. 12.9): At the beginning, the components are placed in the middle in order to get the additional points for the shorter distance to the middle (a). After fields between the middle field and the edge are filled in completely at any point (b).

Then one tries to expand this area further in the radial directions (c). Finally, one tries, taking into account the transition probabilities for the discard, to leave suitable fields free if they cannot be avoided (d). You will find that luck plays a major role here.




(a) Phase 1

(b) Stage 2

(c) Stage 3

(d) Stage 4

Figure 12.9: Filing strategy