universität wien

MASTERARBEIT | MASTER'S THESIS

Titel | Title

Cognitive Functions of an Intelligent Agent in the Context of Problem Solving
On the Example of an Autonomous Forklift Truck

verfasst von | submitted by

Ron Louis Goerz BSc.

angestrebter akademischer Grad | in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien | Vienna, 2024

Studienkennzahl lt. Studienblatt | Degree programme code as it appears on the student record sheet:	UA 066 013
Studienrichtung lt. Studienblatt | Degree programme as it appears on the student record sheet:	Masterstudium Joint-Degree Middle European interdisc. master prog. in Cognitive Science
Betreut von | Supervisor:	Dipl.-Ing. Dr.techn. Paolo Petta

Abstract

The development and deployment of autonomous vehicles in outdoor environments, such as construction sites, require intelligent planning and control systems to tackle emergent obstacles. Achieving a high degree of vehicle autonomy while ensuring appropriate reactive behaviour in dynamically changing environments requires a multitude of cognitive functions.

These AI systems are often developed for specific domains, which means that problem-solving in unfamiliar environments cannot be guaranteed. Issues that appear trivial to humans can pose insurmountable challenges to AI-controlled vehicles. To ensure the safe and responsible use of these technologies, safety in autonomous operation must be guaranteed. This raises the question of how autonomous vehicles can increasingly solve problems independently while simultaneously ensuring safety.

Using the representative example of an automated forklift truck, we implemented a Genetic Learning Algorithm for Behaviour Trees, which we previously tested on our benchmark scenario Pac-Man. The use of Behaviour Trees as a planning and control system offers the advantage of making the decision structure of the autonomous forklift truck readable and comprehensible to humans, thereby easing the assessment of safety aspects. Through the Genetic Learning Algorithm, the forklift truck can learn new Behaviour Trees to solve related but unfamiliar problems. This technical approach delivers positive results in our task scenario as long as we pre-specify the problem. However, the forklift truck needed significant inputs to find solutions to new problems, thereby limiting its autonomy.

In order to improve our understanding of the limitations of the forklift truck, we conducted an interdisciplinary analysis of the Cognitive Functions involved in Problem-Solving. This led us to conclude that the forklift truck lacked the Cognitive Functions of abstraction and comprehension. These Cognitive Functions were taken care of by the designer during specific implementations, which significantly reduced the degree of autonomy of the commercial vehicle in any appreciably different task environment.

Based on this analysis, we suggest corresponding technical solutions to automate the two identified Cognitive Functions, thereby potentially increasing the autonomy of the forklift truck. By embedding these proposed solutions in the structure of the

behaviour trees, safety guarantees can be ensured. In summary, this work contributes to a deeper understanding of AI in the given example by making the Cognitive Functions related to problem-solving explicit, while simultaneously providing an analytical tool for the further development of autonomous systems.

Zusammenfassung

Der Einsatz und die Entwicklung autonomer Nutzfahrzeuge in Außenbereichen, wie beispielsweise auf Baustellen, erfordert intelligente Planungs- und Steuerungssysteme, um aufkommende Probleme zu lösen. Ein höherer Grad an Autonomie eines Fahrzeugs setzt eine Vielzahl kognitiver Funktionen voraus, um ein reaktives Verhalten in einer sich dynamisch verändernden Umgebung zu gewährleisten.

Diese KI-Systeme werden oft für spezifische Umgebungen entwickelt, was dazu führt, dass die Lösung neuer Probleme in unbekannten Umgebungen nicht garantiert werden kann. Problemstellungen, die für den Menschen trivial erscheinen, stellen KI-gesteuerte Nutzfahrzeuge vor teilweise unlösbare Herausforderungen. Gleichzeitig muss die Sicherheit dieser Maschinen für einen potentiellen autonomen Betrieb gewährleistet sein, um einen verantwortungsvollen Einsatz der Technologie sicherzustellen. Daraus ergibt sich die Fragestellung, wie autonome Nutzfahrzeuge zunehmend Probleme selbstständig lösen können, während gleichzeitiger Gewährleistung von Sicherheitsaspekten.

Anhand des repräsentativen Beispiels eines automatisierten Gabelstaplers wurde zunächst ein genetischer Lernalgorithmus für Behavior Trees implementiert, der zuvor am Benchmark-Szenario Pac-Man erprobt wurde. Die Nutzung von Behavior Trees als Planungs- und Steuerungssystem bietet den Vorteil, dass die Entscheidungsstruktur des autonomen Gabelstaplers für den Menschen lesbar und nachvollziehbar ist, wodurch die Gewährleistung von Sicherheitsaspekten ermöglicht wird. Durch den genetischen Lernalgorithmus kann der Gabelstapler selbstständig neue Behavior Trees erlernen, die potenziell unbekannte Probleme lösen. Dieser technische Ansatz liefert gute Ergebnisse. Er erfordert jedoch das Zutun der entwickelnden Person, um Lösungen für neue Probleme zu erlernen, wodurch die Autonomie des Gabelstaplers eingeschränkt wird.

In einem zweiten Schritt wurde der Gabelstapler hinsichtlich der kognitiven Funktionen, die zur Problemlösung erforderlich sind, analysiert. Durch diesen interdisziplinären Ansatz konnte gezeigt werden, dass der Gabelstapler nicht über die kognitiven Funktionen der Abstraktion und des Verständnisses verfügt Diese kognitiven Funktionen wurden im Zuge der Implementierung von der

entwickelnden Person übernommen, wodurch sich der Autonomiegrad des Nutzfahrzeugs signifikant verringert.

Auf Basis dieser Analyse wurden entsprechende technische Lösungsansätze vorgeschlagen, um die beiden identifizierten kognitiven Funktionen zu automatisieren und somit den Autonomiegrad des Gabelstaplers potenziell zu erhöhen. Durch die Einbettung dieser Lösungsvorschläge in die Struktur der Behavior Trees ist es möglich Sicherheitsgarantien zu gewährleistet. Zusammenfassend trägt diese Arbeit dazu bei, das Verständnis von KI zu vertiefen, indem sie die kognitiven Funktionen im Zusammenhang mit Problemlösung im gegebenen Beispiel explizit macht, und bietet gleichzeitig ein Analysetool für die Weiterentwicklung autonomer Systeme.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Paolo Petta, for the numerous discussions, his constant encouragement, and his tireless efforts to support the development of my academic skills.

I extend special thanks to the team at the Complex Dynamical Systems unit at the Austrian Institute of Technology for the fruitful collaboration. My particular thanks goes to Patrik Zips for the supervision at the AIT, and to Johannes Huemer for answering all my questions regarding the forklift truck.

I am grateful to all those who accompanied me during my master's and the writing process: Lilly Eleonor Ripke for the many hours spent together in the library; Maximilian Pößnecker for the stimulating discussions; and Jonas Tiggemann, Paul Grabenberger, and Digvijay Chand for their extensive feedback.

Finally, I would like to thank my family for their lovely support in every situation.

Table of Content

Table of Tables

Table of Figures

List of Abbreviations

AI	Artificial Intelligence
AIT	Austrian Institute of Technology
BT	Behaviour Tree
LRMB	Layered Reference Model of the Brain
MDP	Markov Decision Process
PPA	Postcondition-Precondition-Action

universität
wien

“Intelligence is not a single kind of capacity; it is a diverse collection of capacities to handle the disparate challenges and opportunities that humans encounter in their lives.”

(Daniel C. Dennett)

1 Introduction

Cognitive Science is an interdisciplinary research field, which studies the phenomenon of *cognition* (Abrahamsen & Bechtel, 2012, p. 9). One of the attempts is to approach cognition by simulating it through computer modelling for testing hypotheses (Lieto, 2021, p. 1). On the other side, there is the engineering field of *Artificial Intelligence* (AI), which aims to understand and build *intelligent* machines that behave efficiently and safely in a variety of environments (Russell & Norvig, 2021, p. 1). These two fields of research have interacted with each other since their beginnings (Abrahamsen & Bechtel, 2012, pp. 9–13). Although they differ in their objectives, they are both concerned with similar topics and benefit from each other (Lieto, 2021, pp. 1–2). In this sense, insights in the field of cognitive science inspired developments in AI and vice versa (Russell & Norvig, 2021, p. 3). We take this symbiosis as the starting point of our work.

1.1 Personal Motivation and Background

From a personal point of view, we graduated in Mechanical Engineering and Management at the Technical University of Vienna. Within this study program, we acquired basic skills in maths, classical engineering disciplines and management. During this study, we started taking additional lectures first in political science and later in philosophy at the University of Vienna. Here we focused on logic, theoretical and practical philosophy and specifically on robots and drone ethics. Besides that, we completed an additional curriculum of digital competencies, which equipped us with basic knowledge of computer science and fundamental skills in the programming language Python. Within the Mei:CogSci master's program of Cognitive Science, we realised two student projects. The first one aimed to implement a recurrent neural network for robotic control (Goerz, 2023). The second project was conducted in cooperation with the Jožef Stefan Institute in Ljubljana and developed a learning strategy in the context of industrial robotics (Goerz, Horvat, et al., 2023; Goerz, Simonič, et al., 2023).

Our motivation for the two projects stayed the same for this master's thesis: Based on our background in Mechanical Engineering, we focus on practical solutions for

technical problems. We are interested in insights of Cognitive Science that can inspire practical solutions for technical applications, or formulated from the other perspective: *How can findings in Cognitive Science be practically applied*?
During the process of this master's thesis, we were interested in the difficulties of the transition of theoretical knowledge from Cognitive Science to technical applications. We like to improve our skills in software engineering drawing on our background, and we want to experience what it is like to work on a practical project in an interdisciplinary way. This includes implementing a project technically and reflecting on it theoretically to then conclude further improvements.

1.2 The Example Forklift Truck Scenario

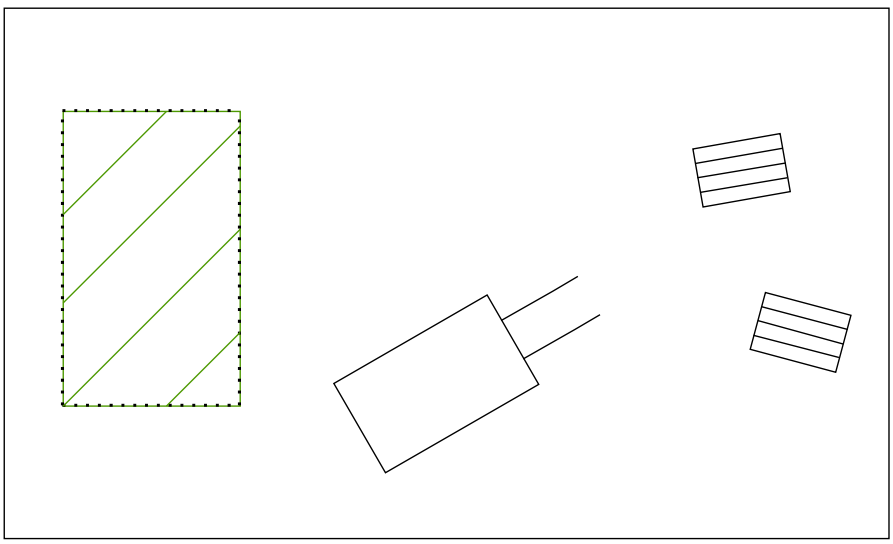


Figure 1: Schematic visualisation of the scenario Crayler

The empty pallets are visible on the right side. On the left side, the defined area (green) for placing the pallets is marked.

Our personal interest matched with an ongoing research project at the Austrian Institute of Technology (AIT), which is a Research and Technology Organisation in Austria. Preceding that, the project "Hopper"[1] aimed to automate the subtask of a remote-controlled forklift truck, also named "Crayler", in an outdoor environment (*HOPPER - AIT Austrian Institute Of Technology*, 2022). As an result, the subtask of the Crayler are automated. However, a control system for autonomous operation

[1] Funded by the "ICT of the future", grant number 873987

was missing. After “Hopper” had been completed successfully, the work was continued by the research engineers Dipl.-Ing. Dr. Patrik Zips and Dipl.-Ing. Johannes Huemer at the unit “Complex Dynamical Systems” at the AIT[2]. In the next step, the goal of the researchers is the *autonomous operation* of the forklift truck. To further develop the vehicle, a task scenario, which we call the *Crayler scenario* in the following, has been defined by the researchers. This task scenario consists of several empty pallets that should be collected and transported to a defined area, which is visualised in Figure 1. To increase the degree of autonomy of the vehicle, a *control structure* is needed that is able to *adapt* to a variety of environments and can capture new knowledge through *learning* (Russell & Norvig, 2021, p. 42).

The researchers at the AIT chose *Behaviour Trees* (BT) as the *method* to control the behaviour of the forklift truck. There exists some evidence in the literature that this method is suitable to control autonomous vehicles in a *domain-general* context, i.e. the transport of goods with autonomous heavy trucks (Colledanchise & Ögren, 2018, pp. 16–17; Iovino et al., 2022). The main properties of this control structure are its *modularity* and its *teleo-reactivity* to changes in the environment (Colledanchise & Ögren, 2018, pp. 23–43). As already mentioned, to increase the degree of autonomy of the vehicle, *learning* is necessary (Russell & Norvig, 2021, p. 42). In a cognitivistic design, the aspect of learning requires augmenting or adapting the *knowledge* captured in the existing control structure with new knowledge gathered through experience (Russell & Norvig, 2021, p. 41). The property of the BTs to be modular makes it possible to add new knowledge by adding additional modules or by recombining existing modules (Colledanchise et al., 2019, p. 188). How the aspect of learning can be realised in the context of BTs is a question of *learning strategy*, which can be examined using the given scenario of the forklift truck. In this sense, there is a need for research and the implementation of a suitable learning strategy for the forklift truck, based on BTs.

[2]https://www.ait.ac.at/ueber-das-ait/center/center-for-vision-automation-control/complex-dynamical-systems (accessed on 30th July 2024)

1.3 Research Question and Procedure

In the previous section, we pointed out that there is a need for the implementation of a learning strategy for BTs in the concrete applied technical scenario “Crayler”. This need is also explicitly mentioned in a survey of BTs, in which Iovino et al. (2022, p. 13) state learning as an open challenge. We were interested whether a BT can be learned in the setting of the concrete example of the forklift truck or not, and if so, to which degree of complexity of the scenario in regard to the autonomy of the vehicle. Dealing with this topic forms the first part of the research question, which is practical-oriented.

Since the overall goal is to achieve a behaviour of the forklift truck that is adaptable to a variety of situations in different environments, we investigate the implemented system theoretically, from the perspective of Cognitive Science concerning its *cognitive functions*. This track has a bidirectional purpose:

On the one hand, it makes the abilities of the implemented system explicit and helps to *understand* it more in-depth. This is necessary because safety guarantees are crucial for the usage of such a vehicle in the real world. According to Iovino et al. (2022, p. 13), BTs are *readable* and *traceable* by a human through their abstract and descriptive structure, which is a prerequisite for safety. However, if mechanically learned, these properties cannot be taken for granted, so a deeper understanding of the chosen learning strategy becomes necessary. Rahwan et al. (2019, p. 478) call for more interdisciplinary research, with the goal of a *better understanding of machine behaviour caused by AI systems*. We assume that investigating our implemented AI system concerning its cognitive functions can contribute to understanding machine behaviour in the concrete case of the forklift truck and can serve as an example for further research. On the other hand, we assume that by identifying the cognitive capabilities of the implemented system, we can recognise the aspects that the system is *not* capable of. In this way, suggestions can be made to guide further research towards the improvement of the forklift truck in the direction of intelligent behaviour.

Through that, the resulting research question has a practical and a theory-oriented part, which both deal with the same subject, namely the forklift truck. We formulate our research question as follows:

RQ: What does a learnable agent program based on the framework of Behaviour Trees for an autonomous forklift truck in the concrete scenario of the Crayler look like, and which cognitive functions are additionally required towards intelligent behaviour in this scenario?

In the following, we introduce and classify key terminology we use, to also draw the disciplinary and thematically boundaries of this work. The first part of the research question is practical-oriented. In that sense, the goal is to design a suitable agent for the Crayler scenario. Russell and Norvig (2021, p. 3) define an *agent* in the context of AI as simply "something that acts", which is deduced from the Latin word *agere* – to *do*. In our example case, the agent is the forklift truck, which is equipped with sensors for perception and can execute certain actions. The mapping from the perceived to an action is defined as the *agent function* and determines the agent's behaviour (Russell & Norvig, 2021, p. 36). The concrete implementation of the agent function is named the *agent program* (Russell & Norvig, 2021, p. 37). So we are interested in a concrete implementation, which is based on BTs. As we have seen before, there is a necessity to integrate the aspect of learning for the autonomous operation of the forklift truck. Russell and Norvig (2021, p. 42) connect the two notions by arguing that an agent that cannot gather knowledge through its perception and a learning process lacks *autonomy*. To answer the first part of the research question, we expect an implemented learning framework capable of learning a BT for the forklift truck in the given Crayler scenario.

The second part of the research question follows a theoretical approach. The used analysis is inspired by methods of *cognitive task analysis*, which are e.g. reviewed by Wei and Salvendy (2004). This method tries to make the cognitive elements of e.g. a job explicit. Within our approach, we prefer the term *cognitive function*, which has been defined in the following way:

> *"Cognitive function is a broad term that refers to mental processes involved in the acquisition of knowledge, manipulation of information, and reasoning." (Kiely, 2014, p. 975)*

The term is used in the discipline of cognitive psychology (Roy, 2013, pp. 448–449). It can also be connected and associated with a transformation from a *task* into an *activity* in a specified context:

> *"By definition, a cognitive function enables its user to transform a (prescribed) task into an activity (effective task). It represents a human cognitive process that has a role in a limited context using a set of resources" (Boy, 1998, p. 266).*

In this sense, cognitive functions can be seen as representing parts of human cognition in specific situations and with that as the building blocks of cognition, accessed through a descriptive approach.

The term "behaviour" is of central importance in this work. It connects our theoretical investigation, which is interested in the question, "*What is 'good' behaviour?*", with on the other side the practical implementation of behaviour, which is concerned with the question, "*How to concretely technically achieve 'good' behaviour?*". The term is defined in a variety of different disciplines, including Biology, Psychology, and Philosophy besides others (Uher, 2016, pp. 478-481). We use the term in connection with the study of the "*Artificial*". Herbert Simon describes this research direction in the following way:

> *"Natural science is knowledge about natural objects and phenomena. We ask whether there cannot also be 'artificial' science – knowledge about artificial objects and phenomena." (Herbert A. Simon, 2019, p. 3)*[3]

Building on this, the authors of the eponymous review define the term *machine behaviour* as "[…] the scientific study of behaviour exhibited by intelligent machines" (Rahwan et al., 2019, pp. 477–478).

[3] First published in 1969.

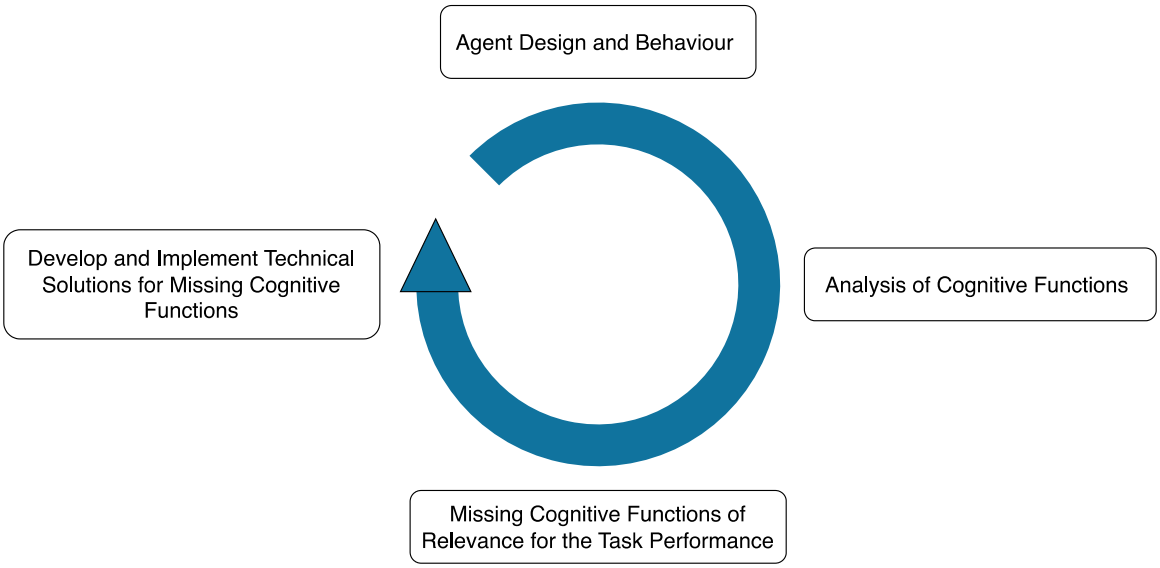


Figure 2: Improvement cycle for the agent design based on the analysis of Cognitive Functions

By incorporating this perspective, our research question contains a *cyclical approach* to improving the agent, which is illustrated in Figure 2. It observes the occurring behaviour of the implemented AI system, including the design process, concerning its cognitive functions, to then analyse them and make further suggestions to improve the system. We assume that by following this cyclic process, it is possible to guide the direction towards the design of machines, which can be called to behave intelligently in connection with their environment.

The term "intelligence" indicates that we are not aiming for an arbitrary behaviour of the forklift truck, but a specific one that meets the requirements for adaptability to different (new) environments and the achievement of goals (Legg & Hutter, 2007, p. 9). Applied to the design of an agent system, Russell and Norvig (2021, p. 4) propose the notion of an “intelligent agent”. In this approach, an agent is intelligent if it chooses rationally the best available action in a specific situation, according to its goal (Russell & Norvig, 2021, p. 34). In that sense, we focus on designing an intelligent agent.

To conclude, in the first step, we aim to implement a learning framework able to create BTs to control the forklift truck in an *autonomous operation mode*. This practical part, covered in Chapters 2, 3, and 4, provides the base for the next step. The second part, covered in Chapter 5, uses the implemented AI system with its learning framework as the object of investigation: Based on the analysis of the

cognitive functions, further suggestions to improve the agent towards an intelligent agent are formulated.

To answer the first part of our research question, we start in Chapter 2 with the definition of a BT that is based on the mathematical concept of a *graph*. We describe suggested design principles and variants of BTs to get familiar with the concept, its properties, and possibilities. At the end of this chapter, we discuss the advantages and disadvantages of BTs from a technical standpoint and give an overview of learning methods in connection with BTs.

Based on Chapter 2, we choose a *Genetic Learning algorithm* as learning method and test it with the *Pac-Man scenario*, an established benchmark example in the field of learning BTs (Zhang, Yao, et al., 2018, p.10). Within this controlled setting, it is possible to make first experiences with the learning method and the agent design. To assess the complexity of Pac-Man's *task environment*, we conduct an analysis of its properties, following Russell and Norvig (2021, pp. 42–47). This enables us to choose a suitable agent type(Russell & Norvig, 2021, pp. 47–58). In the following, we describe the learning framework and compare the learned BTs with a hand-made one to be able to make first statements about the implemented AI system. The lessons learned during the implementation process and the results of the comparison are documented at the end of Chapter 3.

The lessons learned and the components of the learning framework are used for the implementation of a learnable agent program for the Crayler scenario. After describing the scenario and the properties of the agent forklift truck we conduct an analysis of the environment to assess the complexity of the simulation in contrast to a possible real-world scenario and the previous benchmark example Pac-Man. In the next step, we describe the adapted AI system and additional implemented components. The results, which give an answer to the first part of the research question, are summarised at the end of Chapter 4, along with observations and first suggestions for further improvement are described.

In Chapter 5, the second part of the research question is investigated. We first classify the kind of task as a problem-solving task and define the perspective that we take by investigating the respective computational system. We then give a short introduction to cognitive architectures and introduce a model that tries to tackle human cognition through cognitive functions. Following the assumption that our AI system can be inspired by cognitive architectures, we use the introduced model as

a reference to identify those cognitive functions, which are mainly involved in problem-solving, according to the authors of the reference model (Y. Wang & Chiew, 2010). Based on this selection of cognitive functions, we compare them to our AI system and its learning framework. By doing so, we assign the cognitive functions to the system of the forklift truck or the designer partly, depending on their involvement. Through that, it gets possible to understand the AI system more profoundly. At the same time, the cognitive functions, which are assigned to the designer indicate possible domains of improvement. To do this, we examine the concept of "knowledge" from different perspectives, which provides initial insights for the classification of cognitive functions and makes explicit the underlying structure and assumptions on which cognitive functions are built. At the end of the chapter, we identify properties of an AI of relevance in the context of the forklift truck to guide further development on an abstract level. We then conclude our observations, lessons learned from the practical part, and the findings from the analysis of the cognitive functions to make further suggestions for concrete conceptual improvements of the AI system towards intelligent machine behaviour.

2 Theoretical Foundations and Learning of Behaviour Trees

In this chapter, we derive the term “Behaviour Tree” from two perspectives. The focus is on the derivation of a “tree” as a special kind of *graph* in the discipline of mathematics. This is combined with a short introduction of the term “Behaviour”, which we present from the perspective of Biology. We then explain the structural elements, variants and design principles in detail and mention the advantages and disadvantages of BTs. At the end of the chapter, we present possible learning methods in the context of BTs.

2.1 Foundations of Behaviour Trees

To place the term “Behaviour Tree” in the wide range of terminology of different disciplines, we like to give a formal definition at first. Since the applications of graphs, including BTs in AI, spread through various fields of science, we start with its theoretical roots. These can be found in the subfield of mathematics, namely *graph theory* (Kumar & Pattnaik, 2018, pp. 1–6).

2.1.1 From Graphs to Trees

In most of the introductory literature, the hour of birth of graph theory is set to the year 1736 (2003, p. 1; Kumar & Pattnaik, 2018, p. 1). That year, Leonhard Euler wrote an article about the *Konigsberg Bridge Problem*. By using an abstract *model* of the seven bridges and four land masses in the city centre of Konigsberg, he could prove that it is impossible to cross every bridge just once in one walk. Although it is a topological problem, it shows the conceptual properties of a *graph*. The components of a graph $G = (V, E)$ are a set of *vertices* V and a set of *edges* E (Chartrand et al., 2015, p. 3; Kumar & Pattnaik, 2018, p. 2). Vertices are also called *points* or *nodes*, while the terms *lines* or *links* are used for *edges* in the literature (Chartrand et al., 2015, p. 3). One *edge* e connect or join two concrete *vertices* u and v. This can be written as $e = uv$ (Chartrand et al., 2015, p. 3). A

concise definition is given by Chartrand et al. (2015, p. 3): “A graph G is a finite nonempty set V of objects called vertices (the singular is vertex) together with a possibly empty set E of 2-element subsets of V called edges.”

This basic mathematical concept of a graph can be used in multiple specifications in different disciplines. Because of this, we see the necessity to stick to a defined terminology. In the following section, we adapt the used terminology by Chartrand et al. (2015). The *order* of a graph G is defined by the number n of vertices V, while the *size* m is indicated by the number of edges E (Chartrand et al., 2015, p. 4).

We first covered the definition of a graph as an abstract mathematical concept. To make this concept visible and accessible, a suitable *representation* is needed. In the basic literature, two different representations are common. In the former, vertices are represented by points or small circles and edges by lines or curves in a diagram (Chartrand et al., 2015, p. 4; Kumar & Pattnaik, 2018, p. 2). For the purpose of illustrating a BT, the representation in a diagram is slightly changed. Another option is to use *adjacent matrices* for the representation. A graph G with order n can be represented by a $n \times n$ *matrix* $A(G) = [a_{ij}]$, where each element a_{ij} follows equation (1) (Chartrand et al., 2015, p. 39).

$$a_{ij} = \begin{cases} 1 & if \;\; v_i v_j \in E(G) \\ 0 & if \;\; v_i v_j \notin E(G) \end{cases} \qquad (1)$$

Based on the above definition, graphs can be distinguished into several categories. To describe a BT as a special type of graph, we provide a definition of the terms *directed graph* and *rooted tree*.

The difference between a *directed* and an *undirected* graph is the existence of oriented edges. In the literature, directed graphs are also denoted as *digraphs* (Chartrand et al., 2015, p. 161). A directed edge in a digraph connects two vertices with the additional information of orientation. In a diagram, this directedness is represented by an arrowhead on the connection line. In an adjacent matrix, a value different from zero is assigned to just one element a_{ij} to indicate directedness, whereas in an undirected graph, it would also be assigned to the transposed element a_{ji}. We like to mention that bidirectionality is also possible in digraphs.

A *tree* is a special type of graph and can be defined as a *connected acyclic* graph (Chartrand et al., 2015, p. 63; Kaufmann et al., 2003, p. 46). Here, two properties need to be explained. A graph G is connected, if each vertex is directly or indirectly, through other vertices via edges, connected with any other vertex (Chartrand et al., 2015, p. 42). The term “acyclic” indicates that graph G has no *cycles* (Chartrand et al., 2015, p. 63). This means that there is a unique *path* from each vertex u to every other vertex v. A path for undirected graphs is defined through the maximum size of $m = n - 1$ with regard to its order n (Chartrand et al., 2015, p. 5). Connected to that, a cycle is defined by $m = n$ for $n \geq 3$ (Chartrand et al., 2015, p. 5)[4]. A tree, in that sense, can be described as a *simply connected* graph with no cycles. Another access to define a tree is to use the term *bridges*. A bridge in a graph G is an edge $e = uv$ that, if removed, separates G into two disconnected sets, each with at least one vertex (Chartrand et al., 2015, p. 62). In that sense, a tree is a graph, in which every edge is a bridge (Chartrand et al., 2015, p. 63).

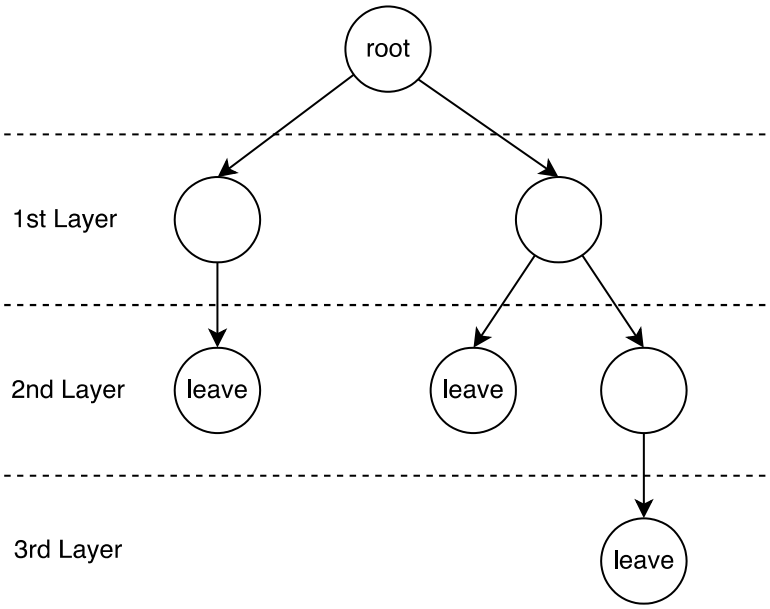


Figure 3: Schematic representation of a directed rooted tree in a layered drawing

Trees are used for different *applications*. They are especially applied in computer science as data structures or as a representation of hierarchies (Kaufmann et al., 2003, p. 46). Since they are used in a wide range of specific cases, there exists

[4] This applies for one cycle, more cycles are also possible and are also defined as a *cyclic* graph.

their terminology and subcategories. The first subdivision to mention is the term *rooted* tree. The root of a tree is a special vertex, which can be seen as an initial and is visualised in Figure 3 (Kaufmann et al., 2003, p. 46). In the representation of a rooted tree as a diagram, the root usually takes a special place, e.g. at the top of the depiction. If the tree is a directed graph, the root is the only vertex, which does not have an incoming edge (Kaufmann et al., 2003, p. 47). Since a tree just has unique paths, there is always a pair of vertices with a directed edge from u to v. In this case, the vertex u is called the *parent* of v and the vertex v is named the *child* of u (Kaufmann et al., 2003, p. 47). If a vertex has no child, it is named a *leaf* (Kaufmann et al., 2003, p. 47). The *depth* of a vertex is defined by the number of vertices on the unique path, beginning from the root (Kaufmann et al., 2003, p. 47). In a diagram, vertices with the same depth can be depicted on the same *layer* as shown in Figure 3. This is named a *layered drawing* (Kaufmann et al., 2003, p. 47).

2.1.2 The Aspect of Behaviour in Behaviour Trees

Besides the technical-mathematical definition of a (specific) tree, there is the second part of the naming “Behaviour Tree”, i.e., the “Behaviour”. This term is defined and used in various disciplines, e.g. Biology, Psychology ,and Philosophy (Uher, 2016, pp. 478-481). From the perspective of Biology, the term is defined in the Oxford Dictionary of Biology as follows:

> *“The sum of the responses of an organism to internal or external stimuli. The behaviour of an animal can be either instinctive or learned.” (Martin & Hine, 2015)*

From this definition, we can extract some aspects and assumptions. A distinction is made between the “internal” and the “external”. In the case of Biology, the “internal” is separated from the “external” by the boundary of the organism. Adapted to the context of AI “external stimuli” are those that the agent can perceive through its sensors from its environment (Russell & Norvig, 2021, p. 36). “Internally”, the agent can keep track of the previous states of the environment and consists of an *agent program* (Russell & Norvig, 2021, p. 37). Depending on these

inputs, the agent selects an *action* (Russell & Norvig, 2021, p. 36)[5]. An agent's action includes e.g. moving from one place to another, as well as inaction (Uher, 2016, p. 479).

In the above definition, "Behaviour" is "the sum of responses" to inputs from the environment and internal signals. The agent's behaviour is in consequence a result of its selected and executed actions or inaction. The term "Behaviour" in "Behaviour Tree" indicates that this specific tree can be used to model this process of responding to stimuli. In the definition is stated that "Behaviour" can be learned. To learn a behaviour in the context of AI using BTs means learning a BT itself, which is what we do in Chapter 3 and 4. The aspect of instinct is not tackled here.

2.1.3 Behaviour Trees as a Special Type of Graphs

In Section 2.1.1, we introduce the basic terminology and the concept of a rooted tree. Together with the notion of "Behaviour", it is possible to provide a definition of a BT. This special type of tree was developed in the game industry to improve non-player characters (Colledanchise & Ögren, 2018, p. 4). Before non-player characters were controlled by BTs, the game industry used *Finite State Machines*, which are difficult to reuse. The key features of BTs, to be *modular* and *reactive,* allowed the game industry to build reusable and better control structures for their non-player characters. These key features are also suitable for other types of agents, e.g. robotics. For a short description of Finite State Machines and Hierarchical Finite State Machines, which can be seen as a predecessor of BTs, the interested reader is referred to Colledanchise and Ögren (2018, pp. 23–33).

As mentioned, a BT can formally be defined as a special type of graph. Our working definition is derived from Colledanchise and Ögren (2018, p. 6) and supplemented with the concept of "Behaviour", derived from Biology:

> Def. 1: *A Behaviour Tree is a directed rooted tree with a hierarchical structure and specialised nodes, which is able to model behaviour.*

[5] In Chapter 5, we provide a definition of the term "rational". Through that, it becomes possible to define a *rational action* that is directed towards an agent's goal.

In the above definition Def. *1*, a directed graph and a rooted tree are implied. Since the concept of BTs has become an academic research subject, a specific terminology for the different elements and functionalities has been established. The leaves of a BT are called *execution nodes*, while the parent vertices are called *control flow nodes* (Colledanchise & Ögren, 2018, p. 6). There are different types of execution and control flow nodes, which are described in the following. Each vertex, including the root, can have one status: *success*, *running* or *failure*, which is returned to its parent (if any) (Colledanchise & Ögren, 2018, p. 6). Thereby, each child is *ticked* by its parent from left to right, so that the embedded functionality is carried out.

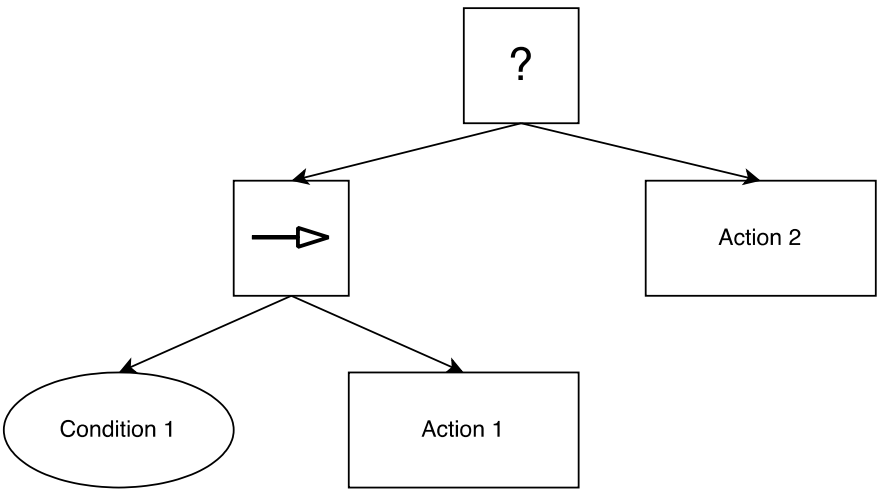


Figure 4: Schematic Behaviour Tree

The BT has a control flow node as a root. On the next layer, there is another control flow node on the left side and an execution node on the right side. The two children of the second control flow node are execution nodes and are located one layer below. Adapted from Colledanchise and Ögren (2018, p. 13, Fig. 1.11).

A schematic BT is shown in Figure 4. It needs to be noticed that a BT has a specific order, which means the arrangement is not arbitrary. In other words, a change of the execution nodes, or parts of the BT results in a different BT and in consequence in a different behaviour.

As we mentioned, there are different types of vertices in BTs. In the subcategory of control flow nodes, there exist four different types in the classical formulation (Colledanchise & Ögren, 2018, pp. 6–8). The following definitions are taken from Colledanchise and Ögren (2022, pp. 6–9):

Types of Control Flow Nodes:

- Sequence:

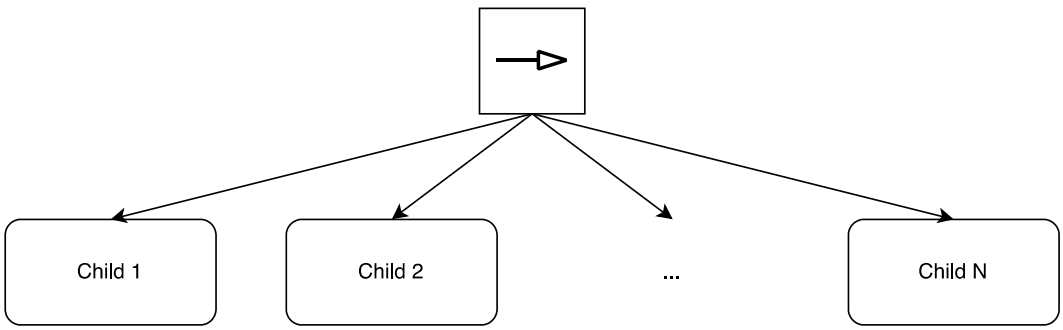


Figure 5: Representation of the control flow node “sequence” and its children Taken from Colledanchise and Ögren (2018, p. 7, Fig. 1.2).

A *Sequence* (see Figure 5) is a special type of a control flow node and is represented by an arrow in a box. It ticks its children, starting from the left (child 1) to the right (child N). If a child is ticked, the child passes its status to the control flow node, its parent. The control flow node only ticks the next child to the right, if the current child returns “success”. If the status of the current child is “running”, also the control flow node adapts this status. In case one child returns “failure”, the next children are not ticked anymore and the parent's status also turns into “failure”. Only, if all children have the status “success”, the status of the control flow node turns into “success”.

- Fallback:

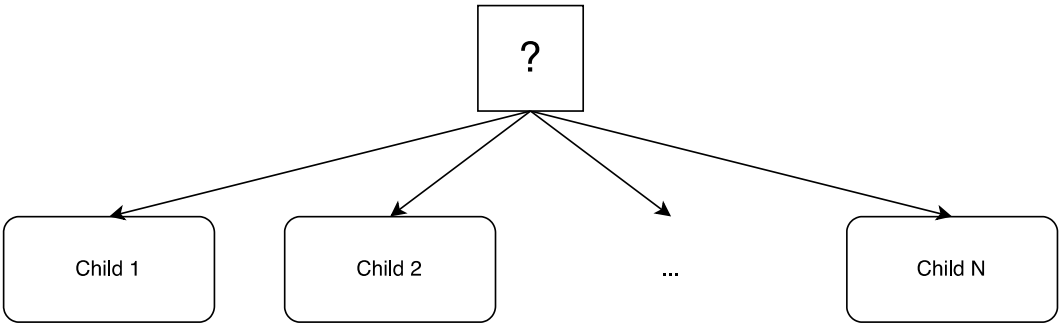


Figure 6: Representation of the control flow node “fallback” and its children Taken from Colledanchise and Ögren (2018, p. 7, Fig. 1.3).

In Figure 6 a *fallback* control flow node is represented. In the literature, this control flow node is also called a *selector* and is illustrated by a question mark in a box. Similar to a sequence, it ticks its children from the left to the right. If a child returns “success”, the fallback node turns its status into “success”. In this case, the remaining children are not ticked. The parent node has the status “running”, if the current child is “running”. If the current child returns “failure”, the next child is ticked. Only, if all children return “failure”, also the status of the control flow node is “failure”. In summary, this control flow node searches for a child, which returns “success”.

- Parallel:

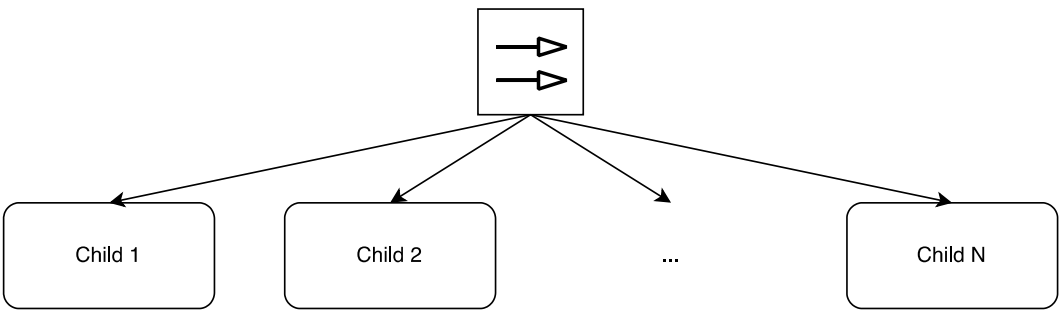


Figure 7: Representation of the control flow node “parallel” and its children Taken from Colledanchise and Ögren (2018, p. 8, Fig. 1.4).

The control flow node *parallel* in Figure 7 is represented by two arrows in a box. This node ticks all its N children simultaneously. If M children return

"success", the state of the control flow node turns into "success". M can be set by a user and it yields $\mathrm{M} \leq \mathrm{N}$. If $\mathrm{N} - \mathrm{M} + 1$ children return "failure", the control flow node turns into "failure". In all other cases, the control flow node parallel has the status "running". This control flow node allows a parallel execution of its children, while the node status can be customised.

- Decorator:

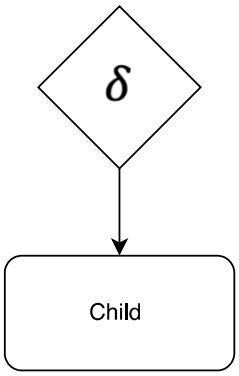


Figure 8: Representation of the control flow node "decorator" and its child
Taken from Colledanchise and Ögren (2018, p. 8, Fig. 1.5).

A *decorator***,** which is schematically shown in Figure 8 is represented by a *policy*, here substituted by δ, and bordered by a rhombus. This control flow node differs from the three previous ones. It only has one child, which is influenced by the decorator. The policy needs to be specified. A policy of a decorator could be to invert the child's return status. Another policy could be that the child is ticked T times in a row. To conclude, the control flow node decorator offers a high degree of customisation.

In addition to the *control flow* nodes, there are also *execution* nodes. In this subcategory, two differentiations exist. The definitions are taken from Colledanchise and Ögren (2018, pp. 8–9):

Types of Execution Nodes:

- Actions:

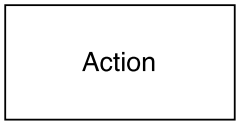


Figure 9: Representation of the execution node “action”

Taken from Colledanchise and Ögren (2018, p. 8, Fig. 1.5).

Action nodes (see Figure 9) are used to execute an embedded instruction. If an *action* is ticked, the node returns “success” if the action is completed correctly. It returns “failure” if the action failed. During the process of carrying out the action the status of the node is “running”.

- Conditions:

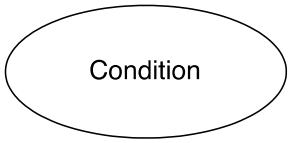


Figure 10: Representation of the execution node “condition”

Taken from Colledanchise and Ögren (2018, p. 8, Fig. 1.5).

The representation of a condition is shown in Figure 10. If this node is ticked, a predefined proposition is checked. If the condition is true, the execution node returns "success"; otherwise, it returns "failure". A condition never returns the status “running”.

With these basic components, it is possible to demonstrate the operating principle of a BT. The BT shown in Figure 4 provides an illustrative example for this purpose. A BT is executed, if the root node is ticked. In the case of the example tree in Figure 4, this root node is the top control flow node. Since this vertex is a selector, it starts ticking the first child on the left side of the diagram. The ticked node is a sequence, in consequence, the first child to the left is ticked. The leaf is “Condition 1”. Regarding the above definition of a condition, the proposition is checked.

Depending on the evaluation, either “success” or “failure” is returned to the sequence. In case the control flow node receives a “success”, it ticks the second child, which is “Action 1”. During the execution of the action, the status “running” is backpropagated to the root via the sequence. In this case, the vertices along the whole path from the root to “Action 1” have the status “running”. This status sustains until the action node returns either “success” or “failure”. If the sequence receives one “failure”, from the condition or from the action, it returns “failure” to the root, according to the definition. Since the root is a selector, it ticks “Action 2” if it receives a “failure” from the sequence. The root has the status “success” if either the sequence or “Action 2” returns “success”. Otherwise, it has the status “failure”, if it is not currently “running”. One execution loop ends, if the root has the status “success” or “failure”. The behaviour of the presented tree can be summarized as follows: based on “Condition 1” the BT checks if “Action 1” or “Action 2” is executed.

2.1.4 Variants and Additional Components of Behaviour Trees

Besides the classical formulation of BTs, there exists a wide range of additional components and variations of basic components. Since the concept is applied to different fields, e.g. game industry or several domains in robotics, slightly different properties are required (Colledanchise & Ögren, 2018, p. 4; Iovino et al., 2022). We describe some additional and adapted components in the following, to demonstrate the wide usability and adaptability of BTs.

- Utility BTs

  As one adaption of BTs, Merrill (2019, pp. 81-89) suggests applying the *Utility Theory* to BTs. This approach aims to include additional information from the environment to select the most suitable action or subtree for the given circumstances. To do so, a selector node is adapted in such a way that its children calculate their expected utility value, which are then compared to each other (Colledanchise & Ögren, 2018, p. 58). For the reason of comparability, this value is normalized within the interval [0,1] (Colledanchise & Ögren, 2018, p. 58). Depending on the expected utility

value, the selector node chooses the highest-ranked action or subtree. The purpose of this adaption is to avoid an a priori static prioritization of single actions or subtrees, which opens the opportunity to assess them continuously while taking additional aspects into account e.g. possible cost of an action (Merrill, 2019, p. 88).

- Stochastic BTs

  Similar to an expected utility value as a decision criterion, Hannaford et al. (2016) suggest using the success probability. In this approach, each child of a control flow node has a specific probability of succeeding. The parent node chooses the child with the highest success probability. Hannaford et al. (2016, pp. 2-3) name this modified selector node “Greedy Selector”, since the procedure follows the concept of a *greedy algorithm*. To assign a probability to a child, the regarding action or subtree is equipped with a specified decorator, which repeats its child until the child succeeds at least once (Hannaford et al., 2016, p. 1). Colledanchise and Ögren (2018, p. 59) notice that the approach runs into an exploit/explore problem and that other categories, e.g. the execution time are not considered.

- Temporary Modifications of BTs

  Sometimes it is wanted that other agents can temporarily influence a BT, without building a new one. The purpose can be to e.g. handle contradictory actions in a BT. Thereby, the controlling agent can be a human or another machine (Colledanchise & Ögren, 2018, pp. 59–62). The idea is that one of the contradictory actions is temporarily hidden in the BT so that just the other action is executed. A switch between the contractionary action is controlled by the agent. The same counts for conditions or whole subtrees. The concept is called *hinted BTs* (Colledanchise & Ögren, 2018, pp. 59–62) and was introduced by Ocio (2012) in the game industry for developers of AI. Another advantage of hinted BTs is that different versions of a BT can be tested dynamically in the process of development.

- Blackboards

  The core idea of using a *blackboard* is to store and retrieve data within a BT that is accessible to all components (Iovino et al., 2022, p. 3; Rovida et al., 2018, pp. 5967-5968). By using the tick of a BT as the timepoint for reading and the moment after the tick for writing on the blackboard, synchronization can be granted. In this way, an execution node is able to retrieve and write the latest data to the blackboard (Rovida et al., 2018, pp. 5967-5968). The blackboard can obtain data regarding the environment or inner states. As a descriptive example, the work of Rovida et al. (2018) can be taken. In their setup, they use two robotic arms for an assembly task. To coordinate and synchronize the two robots, they use blackboards within a BT. The used execution nodes have their own parameters, which are updated in connection with the blackboard. Here the blackboard and the parameter contain data about e.g. the positions of the two robotic arms. The blackboard acts as a platform for synchronised data exchange.

- Active inference

  To enhance reactive action planning in dynamic environments for robots, Pezzato et al. (2022) suggest combining BTs and the concept of *active inference*. This concept was introduced by Karl Friston (2010) and is connected with the theoretical framework of the *free-energy principle*. The paradigm tries to provide a unifying theory for cognitive tasks, e.g. decision-making (Pezzato et al., 2022, p. 2). Based on that, Pezzato et al. (2022, p. 8) suggest including an additional leave node. This node sets a target state, to be achieved by the robot. Active inference and continuous planning are used to select suitable execution nodes. Pezzato et al. (2022, p. 13) claim that a higher robustness can be reached regarding external uncertainties, compared to classical BTs. Also, the number of needed nodes would be reduced, by using their adapted, active inference-based node.

The wide range of possible adaptions and additional nodes in the literature show that the concept of BTs is not closed in itself. Rather, it is adaptable, depending on the use case and the circumstances. It can make sense, to use temporary

modifications in a BT for development reasons or to use a blackboard for the centralized exchange of data within a tree. The framework of BTs allows the integration of abstract paradigms like active inference through its modularity. For sure, this is just an excerpt of published adaptions in the context of BTs. Nevertheless, it points out the possibilities and potentials. At the same time, it shows at which points the concept has its weaknesses and where additional elements with different properties need to be included.

2.1.5 Design Principles of Behaviour Trees

Another aspect of BTs is, how to design them suitable. One of the criteria is human readability. Since the representation of an action node in a diagram mostly includes just a short description of the executed action in a few words, it is not trivial for a human reader to predict the return value. To increase the readability in such cases, Colledanchise and Ögren (2018, pp. 45–46) suggest including conditions, which intercept uncertain cases in such a way that they return “success”. By following this strategy, the readability of BTs can be enhanced.

The structure of the control flow nodes is also not arbitrary. As another criterion, the reactivity of a BT is meaningful. By using sequences in combination with one or more conditions on the left side on the children's level and actions on the right side, an *implicit sequence* can be created (Colledanchise & Ögren, 2018, p. 47). Thereby, the conditions need to be designed as preconditions for the executed actions. The final state can be formalized as a condition, which is put on the left end of the level of the sequences. In this way, reactivity can be increased, as only the necessary and useful actions are carried out depending on the context. (Colledanchise & Ögren, 2018, pp. 46–47).

In some cases, it is more appropriate, to use a structure similar to a *decision tree*. A decision tree is a hierarchically directed tree, similar to a BT (Barros et al., 2015, p. 10). It follows a nested if-then framework of parent nodes to end up at a decision in the form of a leaf (Colledanchise & Ögren, 2018, p. 35). The main difference to BTs is that there is no direct data exchange between the nodes in the form of a “failure” or “success”. It is possible to convert a decision tree in a BT (Colledanchise & Ögren, 2018, pp. 36–37). To do so, the parent node of a decision tree, which

contains a predicate, is converted into a condition in a BT. The structure of the BT follows the one in Figure 4. Here, the two possible leaves of a decision tree are turned into the actions of the BT. In this way, a BT can be designed to own the functional structure of a decision tree.

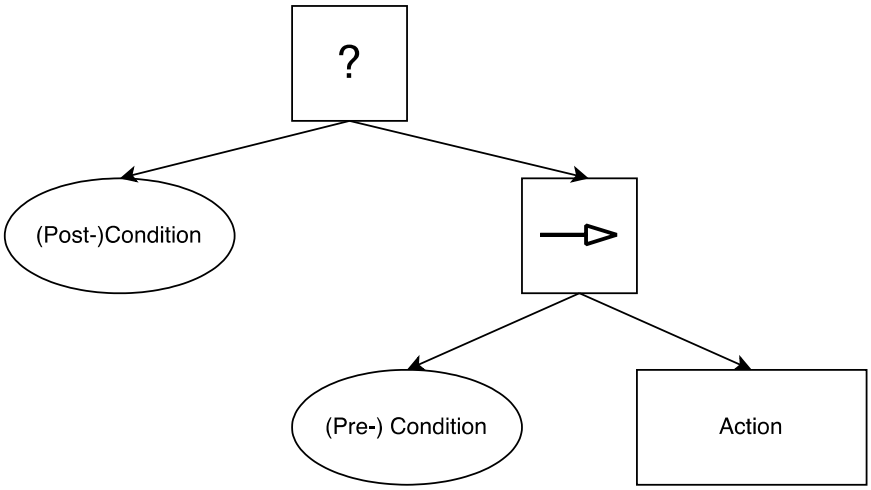


Figure 11: Diagram of Behaviour Tree with a Postcondition-Precondition-Action structure Adapted from Colledanchise and Ögren (2018, p. 49, Fig. 3.8).

Colledanchise and Ögren (2018, pp. 49–51) describe the process of *backchaning* as a way to build *goal-directed* BTs. Thereby, the desired goal is formulated in a condition node, also called *postcondition*, at the first level of the tree structure. To reach that goal, a construction consists of a sequence with a condition and an action as children. The condition on the second level is called *precondition* (Colledanchise & Ögren, 2018, p. 49). Together, the structure is named *Postcondition-Precondition-Action* (PPA) and is shown in Figure 11. By replacing the precondition with a PPA, so that the precondition becomes the postcondition in the new subtree, the BT can be iteratively enhanced and can cover more complex situations (Colledanchise & Ögren, 2018, pp. 49–51).

When working with BTs, it is crucial to determine the appropriate level of *granularity* for each node (Colledanchise & Ögren, 2018, pp. 52–53). This becomes even more important in the context of learning algorithms for BTs, which is described in Section 2.3. Depending on the granularity of the leaves, the tree gets bigger or smaller and with that easier to read for a human or not. If the designer chooses a slim design, one leaf contains a highly abstract description of a condition or action,

while too big designs make the BT confusing through their size. Colledanchise and Ögren (2018, p. 52) propose the following guidelines. If a subtree is always executed in a particular structure, it can be condensed into one leaf. However, if it is likely that parts of the subtree will be used elsewhere in the BT, it should remain as a subtree. It needs to be mentioned that this guideline leaves the question open, when should a leaf be split into a subtree?
These design principles can help to create BTs and can provide guidelines to determine if the size and order of the tree are suitable. This depends on the circumstances in which the BT should be used. An illustrative example, which combines the design principles above, is given by Colledanchise and Ögren (2018, pp. 53–55). If algorithms to learn BTs are used, these principles become even more important.

2.2 Properties of Behaviour Trees

In this section, we introduce some of the main properties of BTs. They are categorised as advantages and disadvantages so that the strengths and weaknesses of BTs are visible. It can be said that there is a consensus in the literature about the positive aspects. However, there are two domains, in which BTs are applied, namely Game AI and Robotic AI (Iovino et al., 2022). As mentioned above, BTs evolved in the game industry and were adapted for technical applications. Nevertheless, the advantages and disadvantages are mostly domain-independent.

2.2.1 Advantages

- Modularity
  As one of the features, the modularity of BTs opens serval possibilities. A BT is modular in the sense that execution nodes, as well as whole subtrees, can be separated independently (Colledanchise & Ögren, 2018, pp. 37–38). These components can be reused in the form of building blocks for new BTs (Florez-Puga et al., 2009). It is mentioned that BTs are more modular if it comes to a higher complexity in contrast to Finite State Machines (Merrill,

2019). This higher degree of modularity results from the fact that the connectivity on each level of a BT is the same (Colledanchise & Ögren, 2018, p. 38; Iovino et al., 2022). On each level, a building block can be inserted, which returns a running, success or failure. Therefore, it doesn't matter at which level the building block is passed. By combining different components with specified properties, a BT with a more complex behaviour can be formed (Champandard & Dunstan, 2019). The property of modularity is a precondition for the use of specific learning strategies, e.g. *genetic algorithms*, which is further explained in Section 2.3.

- Reactivity

  Besides modularity, *reactivity* is often mentioned as one of the main advantages of BTs (Iovino et al., 2022). The term reactivity describes the property of reacting to a dynamic environment. Concretely, it is the ability to switch actions in such a way that the most suitable one is chosen, depending on the current situational status. This status can have an internal or external character, e.g. reacting to a low battery status or dodging an obstacle in the case of a robotic vehicle. In contrast, a static control structure would not adjust its decisions to changed conditions and states.

  In connection with the term reactivity, it is worth mentioning the concept of *teleo-reactive programs*. The notion is introduced by Nils J. Nilsson (1993) and expands the term reactivity with the concept of target orientation, which is indicated by the Greek additive “teleo”. So, an agent pursues one or multiple goals, while it is able to react to changes in the environment. Nilsson (1993) suggested implementing the paradigm in the form of *hierarchical condition action rules*.

  BTs have the ability to integrate the cognitive science-inspired concept of teleo-reactive programs (Colledanchise & Ögren, 2016). This is done by using a specific tree structure combined with continuous root ticking (Colledanchise & Ögren, 2018, pp. 33–35). As a root, a fallback is used. On the first level of the BT, one or several sequence control nodes are used. The children of the sequences are on the left side a condition and on the right side an action. The conditions are formulated in such a way that they

return “success” if the precondition is true. The actions are formulated durative, so that they return “running”, e.g. for the execution of an action called “move forward”. By ticking the tree constantly, the conditions are checked recurringly. By doing so, reactive behaviour can be achieved. The goal orientation is defined by the condition-action-relation, similar to the hierarchical condition-action rules, which are suggested by Nilsson (1993).

- Hierarchical Organisation

  Similar to the idea of teleo-reactive programs, also BTs support a *hierarchical control structure* (Colledanchise & Ögren, 2016). The concept is called *hierarchical organization* or *task hierarchies* in other literature (Colledanchise & Ögren, 2018, p. 38; Iovino et al., 2022). A hierarchical organization has the advantage that a task can be subdivided into several subtasks. Through that, it can be well structured. Iovino et al. (2022) give the example of a game character, which has the task of fighting. In the subtasks of the task, it is decided if a sword or a bow is used with the associated sub-steps. Colledanchise and Ögren (2018, p. 38) draw attention to the possibility of a designer using backchaning in hierarchical control structures, which is described in Section 2.1.5. In BTs, a hierarchical order is realized by its layers or in the more general terms of graphs spoken, its depth. This has the effect that a hierarchical structured tree has a well human readability. But also, it is efficient for a computer to execute a BT, since only the embedded action of one path is executed, except if parallel control nodes or special decorators are used. It needs to be mentioned that efficiency also depends on the used conditions, in case, many computationally intensive conditions need to be checked on one path.

- Information Transport

  Another advantage is the ability to exchange information within a BT. The nodes of a BT are able to communicate with each other through three different responses, as mentioned before (Iovino et al., 2022). Compared to Finite State Machine actions can return the state “running”, which increases the reactivity as mentioned above. Colledanchise and Ögren (2018, p. 5)

state that a BT uses a *two-way control transfer* in contrast to a Finite State Machine, which uses *one-way*. This is realised by the ticking of a node downwards and the reaction of the node that returns information. Besides that, a blackboard can be used for further data transfer. Through that, BTs have an advantage, compared to Finite State Machine with regard to the efficiency of exchanging data within the structure.

- Human Readability and Explainable AI

  Because of their hierarchical tree structure and the support of modularity, BTs are said to have good human readability (Colledanchise & Ögren, 2018, p. 38; Iovino et al., 2022). In comparison to Finite State Machines, they tend to be smaller, while containing the same description of behaviour (Iovino et al., 2022). Iovino et al. (2022, p. 3) state that the hierarchical structure, and with that the subtrees, of a BT follow “a natural level of abstraction”. The advantage of this structure of abstraction becomes visible with the design principle of backchaning, see Section 2.1.5. Here, a task is subdivided, so that even larger trees are well-readable for a human. Colledanchise and Ögren (2018, pp. 16–24) give several examples, in which BTs are used because of their ability to be well human readable. One of them is the project of iQmatic, in which an autonomous truck has the task of industrial applications and is controlled by a BT. With the property of being well readable comes the ability that BTs are easy to design, which facilitates the process of development (Colledanchise & Ögren, 2018, p. 16).

  Since methods like machine learning are more and more applied, the question of explainable AI and safe AI gets more important (Iovino et al., 2022). The property of human readability of BTs can contribute in this regard. A BT can e.g. be learned by using methods of machine learning (Colledanchise & Ögren, 2018, Chapter 8; Iovino et al., 2022). In other cases, subtrees can be learned by human demonstration (Gugliermo et al., 2023). In such approaches, the BT is used as a structured container to represent the learned behaviour. This has the advantage that actions can be controlled and safety guarantees satisfied (Iovino et al., 2022). In other approaches, subtrees are added, e.g. *artificial neural networks*, so that

specific tasks can be learned (Sprague & Ögren, 2022). The advantage of combining these two methods is again that safety guarantees are yield, although stochastic learning was partly used.

2.2.2 Disadvantages

- Complex Implementation

  The implementation of BTs can be challenging and is stated as one of the biggest disadvantages in the literature (Colledanchise & Ögren, 2018, p. 42; Ghzouli et al., 2020). Colledanchise and Ögren (2018, p. 42) point out that the BT engine can be difficult to implement. The reason for this is that the engine must ensure that ticks are generated and travel while the actions are executed in parallel. This is necessary for a well-functional BT. The framework for those engines is mostly prebuilt within a software library, nevertheless, the BT developer needs to deal with it. Ghzouli et al. (2020) observe in their study that BTs are an extensible structure. In this context, they are using the term “language” for BTs and state that it does not have a “proper concrete syntax” (Ghzouli et al., 2020, p. 13). This property is an advantage and a disadvantage at the same time. Its benefit is that the BT can be customised with e.g. special nodes or decorators, see Section 2.1.4. The drawback is that the developer needs sufficient knowledge in language-oriented programming (Ghzouli et al., 2020). Additionally, Colledanchise and Ögren mention (2018, p. 43) that BTs are not focused on states but rather on conditions. In combination with the two-way control transfer, mentioned above, a developer needs some experience in the domain to create efficient BTs.

- Checking of conditions

  Depending on the complexity and size of a BT, it is possible that many different conditions need to be checked (Colledanchise & Ögren, 2018, p. 42). Even if most of the conditions return a “failure” and the corresponding subtree is not executed, it can take some time to receive and process the required data. This disadvantage should be considered during the design

process, since the sample rates of e.g. sensors, in the case of physical vehicles, can be relevant.

- Overfitting
  BTs have a lot of advantages, as stated above. Nevertheless, it is an effort for the designer and it needs computational resources to create and run a BT. If the task environment is rather simple with regard to the six characteristics postulated by Russell and Norvig (2021, Chapter 2.3), the available actions are few, and the task is straightforward, a sequence of condition-action rules can be more appropriate. In those cases, a simpler feed-forward architecture has lower computational and developmental costs (Colledanchise & Ögren, 2018, p. 43).

2.3 Learning Methods for Behaviour Trees

BTs can be created and designed in various ways. In the simplest approach, the concrete tree can be drafted by the developer manually. However, the property of modularity opens up the possibility of applying machine learning techniques. The advantages range from creating whole BTs automatically, to improving them with regard to e.g. their size or performance (Iovino et al., 2022). In other approaches, the tree is modified in such a way that a more adaptive behaviour through (reinforcement) learning can be achieved (Pereira & Engel, 2015). These methods have the disadvantages that they can be error-prone and its implementation can be time-consuming. Iovino et al. (2022) identified four learning methods for BTs in their survey, which are: *Genetic Algorithms*, *Reinforcement Learning*, *Learning from Demonstration*, and *Case-based Reasoning*. Thereby, they included works in the fields of robotics as well as the gaming industry.

2.3.1 Genetic Programming

Learning methods, which use an evolutionary approach form the largest class of research in the field of synthesising BTs, according to Iovino et al. (2021, p. 9, Table 7). The core idea of *Genetic Algorithms* is based on biology and uses the

operations of Darwinian fitness proportionate selection and the recombination of genes (Koza, 1994). The goal of a Genetic Algorithm is to optimise or adapt a given population, although Back (1996, p. 108) argues that there is no difference, except that the two terms originate from different disciplines. The first Genetic Algorithms manipulated fixed-length character strings and were presented by the computer scientist John Holland in 1975 (Koza, 1994). As a further development, *Genetic Programming* was developed as a specialisation of Genetic Algorithms (Colledanchise et al., 2019). By using hierarchical structures of simple functions, e.g. in the form of graphs, the processed complexity is increased (Koza, 1994).

In the context of Genetic Programming, a specific terminology has been established. In the world of Genetic Algorithms, an *individual* represents one specific computer program (Thomas Back, 1996, p. 107). In the special case of Genetic Programming, an individual is a specific graph and applied to BTs, it is one BT with size $E_m > 1$ (Colledanchise et al., 2019, p. 187). A *population* is a set of different individuals. The initial population is randomly generated from the nodes available in the *gene pool* (Iovino et al., 2021). At this point, the property of BTs to be modular is essential.

Once an initial population exists, which is also associated with the term “seeding”, the evolutionary process can start. Each individual is applied to the *task scenario*. The customised *fitness function* assigns a “fitness” as a float to each individual within one *episode*, which is also called a *score* (Colledanchise et al., 2019, p. 188). By designing the fitness function, the developer sets the desired goal, but other aspects like the size and depth of the tree or runtime can be taken into account by specifying *costs* in the fitness function. Depending on the result of the fitness function, called the *fitness value*, the different individuals can be ranked by their “fitness”.

A new *generation* evolves from the initial population, through three genetic operations (Colledanchise et al., 2019, p. 185). By applying *mutation* and *crossover* to the initial *parent* population, new individuals are generated and added to the population. The mutation can cause one of three different modifications (Iovino et al., 2021):

1. Node mutation: one of the nodes of the individual is exchanged with a node from the gene pool.
2. Node addition: an additional node is added to the tree structure at any position.
3. Node deletion: the individual is reduced by a randomly selected node.

For the process of crossover, two parents are needed. Thereby, one single execution node or a whole subtree is spitted and exchanged with a piece of the other parent BT (Colledanchise et al., 2019). In this way, two new offspring BTs evolve, which are different from their parents.
The last of the three genetic operations is *selection*. This mechanism decides, which individuals are kept in the population (Iovino et al., 2021). To do so, there are different types of selection strategies (Iovino et al., 2021):

- Elitism: the individuals with the highest score are kept in the population.
- Tournament Selection: two individuals compete against each other, so that the one with the higher score, of the two, is kept in the population.
- Rank Selection: if an individual is kept in the population depends on a probability, which is proportional to its score.

As described by Iovino et al. (2021), the genetic operation of selection is at least applied two times during one evolutionary iteration. At the first time, it selects parents for the mutation and/or crossover. After the new individuals are added to the population, they are evaluated by the fitness function. The survivors of another selection process of this population form the new generation and are the *offspring* of the parents (Iovino et al., 2021).

In the following process, the above procedure is iterated until a desired score or a pre-set amount of generations is reached. Depending on the simulation time, thousands of generations are possible as in the approaches of Nicolau et al. (2017), to name one of many examples. It needs to be mentioned that the applications, in which Genetic Programming is used are customised. Software packages like those used in Section 3.2.3 come with a lot of configurable parameters. Some approaches, like the one presented by Colledanchise and Ögren (2018, pp. 139–149), use combinations of Genetic Programming and other methods, e.g. a greedy algorithm. More advanced methods in the field can be looked up in the guiding book by Poli et al. (2008).

2.3.2 Reinforcement Learning

In this Section, one of many approaches in connection with *Reinforcement Learning* is presented. The proceeding is suggested by Colledanchise and Ögren (2018, Chapter 8.3) for a combination of Reinforcement Learning and BTs and uses the often-used method of *Q-Learning*. The technique is capable of solving a class of problems, which can be described by a *Markov Decision Process* (MDP). To understand the mechanism of Q-Learning, it is necessary to give a short introduction to MDP.

An MDP can be seen as a model for a discrete decision process and is described by Sutton et al. (1999). It consists of a tuple of four sets (S , A , p , r) (Sutton et al., 1999). Thereby, $s_t \in S$ is the perceived state of an agent in an environment at timepoint t. $a_t \in A_s$ is the executed action at time t, while a finite, not empty set of actions is available at a specific state s_t (Sutton et al., 1999). By performing an action a_t, a new state s_{t+1} can be reached with a transition probability $P(s'|s,a) = P(s_{t+1} = s'|s_t = s, a_t = a)$ (Sutton et al., 1999, p. 184). Therefore, the agent receives a reward $R_a(s,s')$ (Sutton et al., 1999). The goal is, to learn a policy $\pi: S \rightarrow A$, which maximises the expected long-term reward (Sutton et al., 1999).

For a given MDP problem, the Machine Learning technique Q-Learning can be used to optimise the expected long-term reward. To do so, the so-called *Q-function* $Q: S \times A \rightarrow \mathbb{R}$ is used, which assigns the expected reward to each combination of state and action (Colledanchise & Ögren, 2018, p. 150). During the learning

process, the Q-function is updated according to equation (*2*), given by Pereira and Engel (2015, p. 6, Equation 5):

$$Q_{k+1}(s,a) = (1-\alpha_k)Q_k(s,a) + \alpha_k[r + \gamma \max_{a' \in A_{s'}} Q_k(s',a')]. \qquad (2)$$

In each iterative step k, the corresponding state-action pair is updated with the received reward. Thereby α_k is a learning rate parameter which varies over time (Pereira & Engel, 2015). The apostrophe indicates the next state, while $\gamma \in [0,1]$ is a discount-rate parameter. It can be shown that Q_k converges to the optimal value Q^*. The interested reader is referred to Pereira and Engel (2015) for a detailed description of the algorithm.

The application to BTs is done by replacing special *learning nodes* (Colledanchise & Ögren, 2018, p. 151; Pereira & Engel, 2015). These learning nodes can be action nodes or control flow nodes in the form of fallbacks. In a learning action node the Q-Learning algorithm is encapsulated, including states, actions and rewards (Colledanchise & Ögren, 2018, p. 151). In the case of a learning fallback node, the possible actions are the children of this control flow node (Colledanchise & Ögren, 2018, p. 151). Depending on the current state, the control flow node chooses the child with the highest expected reward, depending on the task (Colledanchise & Ögren, 2018, p. 151). This idea is similar to the extension of stochastic nodes with the advantage that continues optimisation is possible.

Other approaches, which try to combine Reinforcement Learning and BTs are using similar techniques (Dey & Child, 2013; Fu et al., 2016). The goals range from applications of Q-learning for choosing actions, to enhance design decisions of BTs. Banerjee (2018) even goes a step further and uses Reinforcement Learning to create BTs autonomously via control policies in the setting of repeated task interaction.

2.3.3 Learning from Demonstration

The core idea of *Learning from Demonstration* is that a person can demonstrate a particular behaviour and does not need to write the corresponding program (Iovino

et al., 2022). Once the underlying infrastructure is implemented, the advantages are that less time and less knowledge are required to learn a particular behaviour, compared to implementing it manually. By applying the method to BTs, the demonstrated behaviour is stored in subtrees (Colledanchise & Ögren, 2018, p. 154). Besides storing a specific behaviour in a structure, it has the benefit that others can read the BT. Colledanchise and Ögren (2018, p. 154) note that the proceeding results in large BTs. To avoid large BTs, Robertson and Watson (2015) suggest a method, which is capable of finding common patterns and tries to reduce the size through generalization in an iterative process.

2.3.4 Case Based Reasoning

This approach tries to apply behaviour, or parts of it, from already trained and solved problems to new problems (Iovino et al., 2022). Again, the property of BTs to be modular provides a suitable framework for this machine-learning technique. Iovino et al. (2022) note that the method has a drawback. It is difficult to refine a learned problem-solving strategy. However, a specific Reinforcement Learning approach can cushion this disadvantage.

3 Implementation of the Benchmark Example Pac-Man

In this chapter, we describe the implementation of a Genetic Programming Algorithm for our benchmark example Pac-Man. Based on the observations during the process of implementation and a comparison between a hand-made BT and learned BTs, we gather first insights about the learning method and can use the implemented framework, together with lessons learned for the implementation of the forklift truck scenario.

3.1 Pac-Man as a Benchmark Example in AI

To test and evaluate learning methods in AI, observable and controllable examples are useful. If those setups are used for different approaches, they become comparable. In the context of BTs, Pac-Man is used as a benchmark example for the learning approach of Genetic Programming (Zhang, Yao, et al., 2018). Besides Pac-Man, other platform games like the Mario AI are established as a benchmark (Iovino et al., 2022; Nicolau et al., 2017). We like to mention that different versions of the game Pac-Man exist. Colledanchise and Ögren (2018, pp. 11–14) present a version that is comparatively simple, through its smaller game board and the low number of enemies. Compared to that, Zhang et al. (2018) use another version of the game, called Ms. Pac-Man, which is similar to the original Pac-Man game. We used a modification of the original version of Pac-Man provided by Leamy (2020).

3.1.1 The Properties of the Agent

The agent in the game is called *Pac-Man* and appears on the *game board* as a yellow circle with a moving opening, which is suggested to be a mouth. Its goal is to score, by collecting *pills* on the game board or attacking its enemies. There exist *small pills* and *power pills*, also called *big pills*. The latter has the property of changing the state of Pac-Man's enemies so that they can be attacked by the agent

for a short period of time. If Pac-Man’s enemies are not in this state, the agent can be attacked by them.

This leads to the agent's primary actions and perception. Pac-Man is capable of moving in four directions on the two-dimensional game board. These are: *up*, *down*, *left and right*. It can observe its environment within three fields around itself. Additionally, it knows the location of the four power pills and receives the current mode of its enemies.

We try to design the agent in such a way that it is similar to what the forklift truck is capable of. This counts for the perception and the available information about the environment. Since the forklift truck can only observe its surrounding area, we designed the agent in the Pac-Man scenario in the same way. The information about the position of the power pills draws a parallel to the known position of the pallets, which the forklift truck should load. The capabilities of the forklift truck are listed in Section 4.1.1.

3.1.2 Pac-Man's Environment

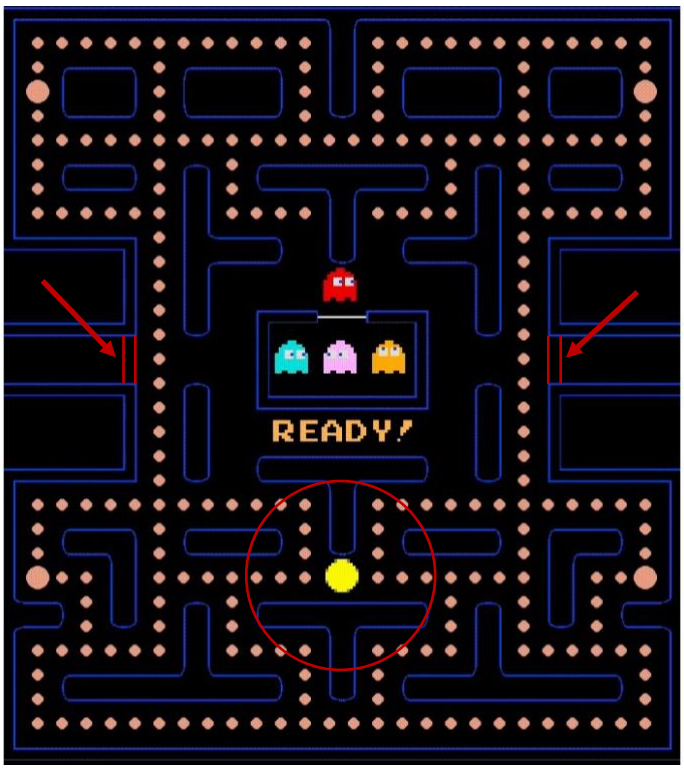


Figure 12: Adapted game board of our Pac-Man benchmark example

Visible are the four ghosts (enemies with colouring: red, cyan, pink, and orange) and the representation of the agent (Pac-Man with yellow colouring) with an observation radius (red circle). We modified the maze (blue border) by two red walls, which are indicated by the two red arrows. The small pills are the smaller light orange circles, and the bigger ones are the power pills.

The game board is visible in Figure 12 and shows the situation at the beginning of one game. The blue lines on the board indicate the shape of the maze, in which the different agents can move. In the middle of the game board, a box, also called the "nest", is located, in which the four enemies start and get reanimated (Galvan-Lopez et al., 2010). Pac-Man is not able to enter this area. The agents in the original maze have the opportunity to leave the game board on one edge in the middle of the horizontal direction, to get on the opposite edge back in the game. To simplify the structure of the maze, we disabled the feature by the insertion of additional walls at the entries of the tunnels. This is indicated by the red lines in Figure 12.

Table 1: Table of rewarded points in our modified Pac-Man benchmark scenario

Item	**Reward**
Small pill	10 Points
Power pill	50 Points
1ST Ghost	200 Points
2nd Ghost	400 Points
3rd Ghost	800 Points
4th Ghost	1200 Points

In the corridors of the maze, pills are placed. As mentioned, Pac-Man is able to “chomp” those pills and receive points by moving to the respective location. For each small pill, Pac-Man receives 10 Points. If a power pill is chomped, Pac-Man receives 50 Points and the ghosts turn into *scared mode*. During this time span, the ghosts can be chased by Pac-Man. The reward for chomped ghosts within one slot is staggered and is shown in Table 1. Additional pellets in the form of fruits occur in the original game. To simplify the environment of Pac-Man and make sure that the agent does not get points by accident, additional pellets, which occur in the original game are reduced.

The enemies of Pac-Man are the four *ghosts*. Each of them performs different actions with the goal of catching Pac-Man. If Pac-Man is caught by a ghost, we modified the game in such a way that it is over. In the original version, a player has three lives and can reach the next level, each time all pills are chomped.

The movement of the agents happens in discrete steps. If the ghosts are not in the scared mode, both, Pac-Man and its enemies have the same speed. In the case of scared mode, the ghosts move with a fraction of Pac-Man's speed.

3.1.3 Complexity Analysis of the Pac-Man Benchmark Task Environment

As suggested by Russell and Norvig (2021, p. 60), the first step towards designing an agent is to specify the *task environment.* This includes the *external environment* of the agent, as well as its *actuators*, *sensors,* and the *performance measure*.

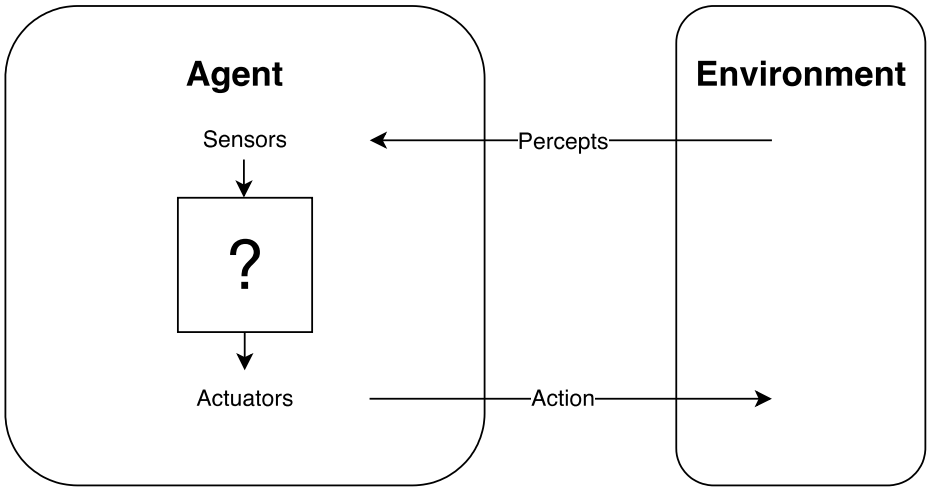


Figure 13: Schematic illustration of an agent and the interaction with its environment.

The box with the question mark represents the agent program and can be filled with e.g. a BT. Adapted from Russell and Norvig (2021, p. 37, Fig 2.1)

To illustrate the connection between the agent and its environment, Russell and Norvig (2021, p. 37, Fig 2.1) suggest the representation shown in Figure 13. We like to point out that in this illustration the agent is situated in an action-perception loop through the interaction with its environment. Through this perspective, the environment can be thought of as a part of the agent's decision process, similar to the idea of *active externalism* (Clark & Chalmers, 1998). A closer view is taken in sub-section 5.1.1, whereas at this point we focus on the technical design of the agent.

Applied to the example of Pac-Man, the external environment is the game board, including the four ghosts and the different pills. The actuation of the agent Pac-Man in the virtual setting is to prompt the direction of movement. In the task environment, the agent’s perceiving is included. In the case of Pac-Man, this is the perception of its closer area (red circle in Figure 12), as well as the mode of the ghosts and the information about the position of the power pills. The performance measure is the score since the agent's aims to maximise it. Based on this specification, Russell and Norvig (2021, pp. 43–47) suggest six properties to categorise the task environment. We analyse the task environment in the following:

- Observability:

  There exist three alternatives of observability, which are *fully observable*, *partially observable* and *unobservable* (Russell & Norvig, 2021, pp. 43–44). A task environment is fully observable if the agent perceives all relevant aspects to choose an action through its sensors, according to the performance measure. In the case of unobservability, the agent does not perceive anything through its sensors. According to Russell and Norvig (2021, pp. 43–44), a partially observable environment is defined as being neither fully observable nor unobservable. That means that not all relevant aspects are observable by the agent, according to the performance measure.

  In the example of Pac-Man, the agent perceives its closer environment, but not the whole game board. Through that, it does not know the position of its enemies nor of the remaining small pills. These aspects are score-relevant so we deduce that the task environment is partially observable.

- Number of Agents:

  The authors distinguish between *single-agent* and *multi-agent* task environments (Russell & Norvig, 2021, pp. 44–45). If another entity *B* is categorised as another agent or as an obstacle depends on its behaviour. If the goal of its behaviour can be best described as a maximisation of a performance measure, which is related to agent *A*, then *B* is categorised as an agent.

  Applied to the setting of Pac-Man, the four ghosts can be categorised as agents. As mentioned before, each ghost has a different behaviour. This behaviour can be described in a performance measure, which is maximised by catching Pac-Man, while not getting chomped in scared mode. In this sense, it is a multi-agent task environment.

- Determinacy:

  According to Russell and Norvig (2021, p. 45), the task environment can be *deterministic* or *nondeterministic*. If the next state of the environment only depends on the current state and the actions of the agent(s) in a

deterministic way, then the task environment is deterministic. It is nondeterministic if the next state also depends on other factors. The agents can be cooperative or competitive. The important point is that their actions cause the environment deterministically. It is also possible that stochastic elements are included in the environment. Then, it is called a *stochastic* environment.

In the case of Pac-Man, all actions result in defined changes in the environment by executing primary actions. Also, the actions of the ghosts cause deterministic changes. The next state of the environment only depends on the (deterministic) actions of the agent and the current state, so that it is deterministic.

- Episodic or Sequential:

  A task environment can be *episodic* or *sequential* (Russell & Norvig, 2021, p. 45). It is episodic if the agent's situatedness can be distinguished in atomic episodes, which are not influenced by its previous actions. In sequential environments, the actions of the agent affect its future decision, e.g. through a changed position.

  The Pac-Man environment is clearly sequential since the agent changes its position at every step. Through that, the decision in the following step depends on the new environmental situation, which is caused by the previous action.

- Static or Dynamic:

  If the environment is changing, while an agent is deciding on an action, it is called *dynamic* (Russell & Norvig, 2021, pp. 45–46). In case, the environment is not changing during the time of deliberation, but the score of the performance measure is, then the task environment is *semidynamic*. It is *static* if the environment and the score do not change before an action is selected and executed.

  While Pac-Man is deciding on an action, the ghosts are moving. Even if the agent Pac-Man does not move at all, its environment changes, caused by

the ghosts. Since the environment changes while the agent is deliberating, it is dynamic.

- Discrete or Continuous:
  Russell and Norvig (2021, p. 46) differentiate between *discrete* and *continuous* environments. The decisive factor is, how are states of the environment are handled over time. In the case that the environment changes in separate steps, it is discrete. Otherwise, it is continuous.
  In the virtual example of Pac-Man, each agent moves simultaneously at a specific tick. Through that, the environment changes in separate steps. According to the above categorisation, the environment is discrete.

- Known or Unknown
  Russell and Norvig (2021, p. 46) name another category, which refers not only to the environment but also to the designer of the agent. The distinction is between a *known* and *unknown* environment. It is known if the designer and the designed agent are able to foresee all outcomes of the agent's actions. The authors note that this category is not dependent on observability, since it refers to the agent's and designer's knowledge about how the environment functions. Russell and Norvig (2021, p. 46) describe this as the "laws of physics" in the environment.
  In the Pac-Man example, it needs to be distinguished between the primary actions, which are moving in one direction, and *secondary actions*. These secondary actions are based on the primary actions and are extended by additional data about the environment. An example is the action "avoid ghost", which is used in the implemented BT and described in section 3.2.4. Here, the rule of getting caught by a ghost is inherent in the action, which is given through the design. That indicates that the environment is rather known than unknown.

Table 2: Categorisation of the task environment in our Pac-Man benchmark scenario

Low Complexity	Pac-Man Environment	High Complexity
fully observable ↘	partially observable →	unobservable ↗
single-agent ↘	multi-agent ↗	multi-agent ↗
deterministic ↘	deterministic ↘	nondeterministic ↗
episodic ↘	sequential ↗	sequential ↗
static ↘	dynamic ↗	dynamic ↗
discrete ↘	discrete ↘	continuous ↗
known ↘	known ↘	unknown ↗

Categorisation between an environment with the lowest (green and symbol (↘)) and highest (red and symbol (↗)) possible complexity. The colour yellow with the symbol (→) indicates an in between. Adapted from Russell and Norvig (2021, p. 47, fig. 2.6).

By following the categorisation by Russell and Norvig (2021, pp. 43–47) we can be assess the task environment. The authors describe that the most complex categorisation is an unobservable, multiagent, nondeterministic, sequential, dynamic, continuous and unknown task environment. These complex conditions require an appropriate agent design. The opposite of this categorisation describes a simpler environment as presented in Table 2. In that case, simpler kinds of agents can be used, like the *simple reflex agent*, which is based on condition-action rules (Russell & Norvig, 2021, pp. 49–51). Besides the simple reflex agent, the authors name the *model-based reflex agent*, the *goal-based agent* and *the utility-based agent*. Each of them differs concerning their structure and can be implemented as a *learning agent* (Russell & Norvig, 2021, pp. 49–51).

To conclude, the task environment of Pac-Man is partially observable, multi-agent, deterministic, sequential, dynamic, discrete and known. In Table 2 it is visible that the task environment has a middle degree of complexity. Three up to four of the seven categories can be classified as more complex. In that sense, the environment is not simple enough to use a simple reflex agent. The fact that the environment is partially observable indicates the need to keep track of the unobservable parts of the environment through an inner model. According to Russell and Norvig (2021, pp. 51–53), a *model-based agent* can keep track of

previous states of the environment. This property makes it suitable for dealing with partially observable task environments. A *goal-based agent* chooses an action that fits its *goals* (Russell & Norvig, 2021, p. 54). A goal in this sense has a binary character. A specific state is a *goal state* or not, but the agent does not distinguish between “better” or “worser” goals regarding e.g. their costs (Russell & Norvig, 2021, p. 54). If the agent is designed as a *utility-based agent* it chooses the action that has the “best” *expected utility*, which is a computed non-binary value (Russell & Norvig, 2021, pp. 54–55). However, these agent structures are more complex and need more resources for implementation compared to simpler agent designs like a model-based agent. For a closer view, the interested reader is referred to Russell and Norvig (2021, pp. 47–60, chap. 2).

3.2 Implementation Structure of our Pac-Man Benchmark Scenario

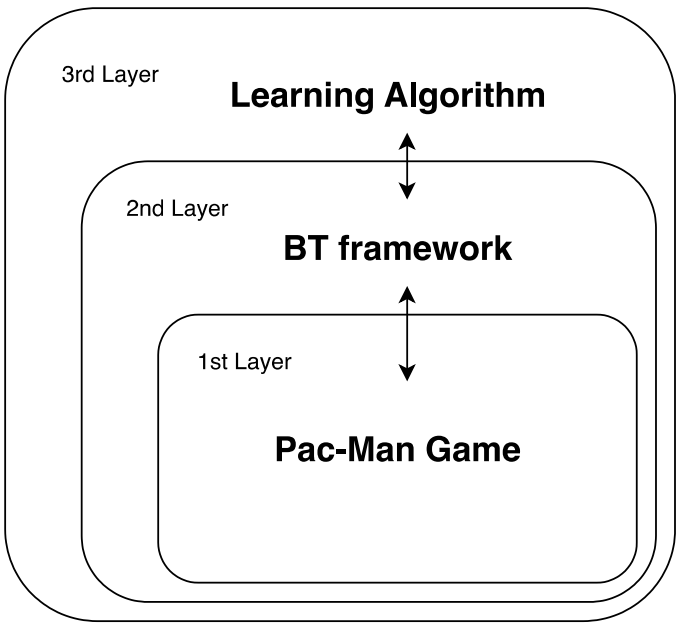


Figure 14: Three-layered structure of our implemented Pac-Man Benchmark Scenario

The implementation of our Pac-Man benchmark scenario follows a three-layered structure, which is shown in Figure 14. The order of the layers also represents the temporal proceeding. During the process of implementation, we integrate the different packages into each other, so that an embedded structure evolves. In the first step, we use an open-source implementation of the game Pac-Man and adapt

it for our purposes. Based on that, the second layer contains an implementation framework for BTs. To learn different BTs with a suitable learning method, an algorithmic structure is used on the third layer.
To select suitable open-source libraries, we formulate the following selection criteria:

- Integrability: It needs to be possible to integrate the software of the different layers into each other.
- Programming Language: The algorithms should be implemented in the programming language Python.
- Actuality: The software should be up to date, which means that the last commit should not be older than 2020.
- Adaptability: The frameworks should not be implemented as closed systems, but more open for adaptions and additional components.

These criteria function as an orientation to select suitable software implementation. On each layer, additional criteria come into account, regarding their specific requirements. To be able to reproduce and comprehend the implementation process, the main aspects are described in the following.

3.2.1 Used Pac-Man Implementation

We select an existing Pac-Man implementation that we can adapt for our purposes to reduce the programming effort. It is implemented in the programming language Python, is not older than 2020 and can easily be adapted and integrated. The latter provides another criterion for the selected implementation, because it needs to be adaptable with regard to the control of the agent Pac-Man and other game-specific parameters. Compared to the implementation suggested by Colledanchise and Ögren (2018, pp. 11–14), the game board shown in Figure 12 provides a higher degree of complexity through the larger number of ghosts (4 vs. 2) and the bigger size (28x31 vs. 20x7).
The implementation that is used by Colledanchise and Ögren (2018, pp. 11–14) has the advantage that it includes a framework to integrate BTs. We decided

against this software because it is partly implemented in the programming language C++ and does not fulfil the required criteria. As a result of a search on the developer platform “GitHub”, we found an implementation of the Pac-Man game by Devin Leamy (2020). The developer implemented the game for the purpose of practice in the programming language Python. However, it fulfils our criteria and comes with an implementation of the original game. In his implementation, the agent Pac-Man is controlled by a human player through a keyboard. As a framework for running the game, the latest version (2.3.0) of the Python-specific software package Pygame is used. The package provides a set of functionalities for game design, e.g. the possibility to create a *Graphical User Interface* (Andrews et al., 2023). An integrated clock controls the speed of the game and allows the synchronization of different agents.

To provide a framework with similar controlled conditions and able to be controlled by a BT, we modified the software of Leamy (2020). In the first step, we modified the game conditions. To reduce the time for running one game, we decided that the agent has just one life instead of three. Under these conditions, it is difficult for an untrained human player to reach the second level of the game. So, we restricted the setting to the first level. During the process of training, no AI was able to collect all the pills, which shows that this border was not touched. As mentioned before, we closed the tunnels on the two sides of the game board, so that no agent can pass them. We did this to reduce the implementation effort with regard to the area Pac-Man is allowed to observe.

In our implementation, the game board is represented by a 28 across 31 tall matrix. The integer values of the matrix indicate the different pills, a wall or an empty field. Through that, the board is separated into distinct boxes. One clock tick during the game moves the representation of an agent about a quarter box, through which a smooth visualization is realised. It also opens up the possibility to give a prompt every quarter of a box and not just at each box, which allows for faster reaction.

We implemented the control of the agent Pac-Man through a function from the package “Pygame” (Andrews et al., 2023). By adapting the regarding section, the agent could be controlled by a sequence of primary actions. This provides the base for controlling the agent Pac-Man through BTs.

3.2.2 The Behaviour Tree Framework

Colledanchise and Ögren (2018, p. 42) note that the implementation of BTs can be difficult. Especially, the BT engine, which needs to guarantee a smooth process is crucial. Because of that reason, an existing software package is used. Iovino et al. (2022) compared different open-source libraries for BTs. According to their evaluation, the software package “py_trees” (Stonier, 2023) fulfils the above-mentioned selection criteria as the only one. The library is implemented in the programming language Python and is up to date. Besides that, there exist libraries for Genetic Programming, which include the named software package. In that sense, we selected the library “py_trees” for the implementation of BTs. The software offers the opportunity to use control flow nodes, blackboards and customised execution nodes. Besides that, it is also possible to include decorators. The library provides a skeleton to implement execution nodes. An example is given in the documentation (Stonier, 2020). It can be specified, at which point in time a particular part of the program is executed. In the case of the Pac-Man implementation, the relevant section for an action or condition is only executed, if it is ticked by its parent. We connected the two layers of the implementation through a dictionary, which functions as a container. This container includes data about the game status, e.g. if the ghosts are scared or not. At each *game tick*[6] the root of the regarding BT is ticked. At the level of the execution nodes, the necessary data are received from the container. In the case of an action node, the selected primary action is passed to the container. After an action is selected in the BT, the game proceeds for one game tick. The new game status is passed to the container and the next game tick starts.

3.2.3 The Learning Framework

It is possible to learn BTs with several methods. According to Iovino et al. (2022), the approaches can be separated into four different strategies, which we described in section 2.3. As we mentioned in section 2.3.1, researchers use Genetic

[6] There exist two different types of “ticks“. The first one presented in this work is the tick in connection with BTs. This is the impulse, initialized by a parent node or from the outside, to execute a node in the BT. The second type of tick is related to “pygame” and controls the sequence of the program. This tick is named “game tick”.

Algorithms for automated BT design. Especially, the similar works of Iovino et al. (2021) and Zhang et al. (2018) are worth mentioning. The latter uses a similar Ms. Pac-Man task environment to our scenario, while the first uses a simulation of an industrial-robotic application. In both cases, Genetic Programming is used to learn BTs. Iovino et al. (2021) argue that the modular structure of BTs fits well for Genetic Programming frameworks. The results of this automated way of designing generate robust and reactive BTs with regard to the used training environment (Iovino et al., 2021). Iovino et al. (2021) mention that Genetic Programming suits well for learning tasks in virtual environments and applying them then in the real world. This is because many evaluation iterations are necessary for the learning process, which can be conducted in less time in a simulation compared to the real world. Since the learning framework is also tested in industrial-robotic applications, it seems promising to us to use Genetic Programming for the forklift truck scenario as well. In this sense, we decided to use Genetic Programming, since the method can be applied to the benchmark example and the scenario of the forklift truck. The literature mentioned above indicates that the method suits both cases.

We integrated a distribution for the implementation of a Genetic Programming algorithm that is based on the work of Iovino et al. (2021). We use the distribution that is provided by Escudero (2022) and adapted it for our purposes. For the implementation, the programming language Python is used by the developers. Genetic operations on BTs are performed by operating on a list. These lists consist of strings that represent the nodes of a BT. Through a software interface, these lists of strings are connected to the “py_trees” framework. Within “py_trees” each BT of one Generation is executed during one iteration. We describe the principle proceeding of the Genetic Programming algorithm in Section 2.3.1. As visible in Figure 14, the Pac-Man game is embedded within the third layer. This is done by using the provided framework for implementing a customised scenario, which is called “world” in the used repository. To summarise at this point, the Pac-Man game is used as the *task scenario* for the different BTs, which are executed by using the “py_trees” framework.

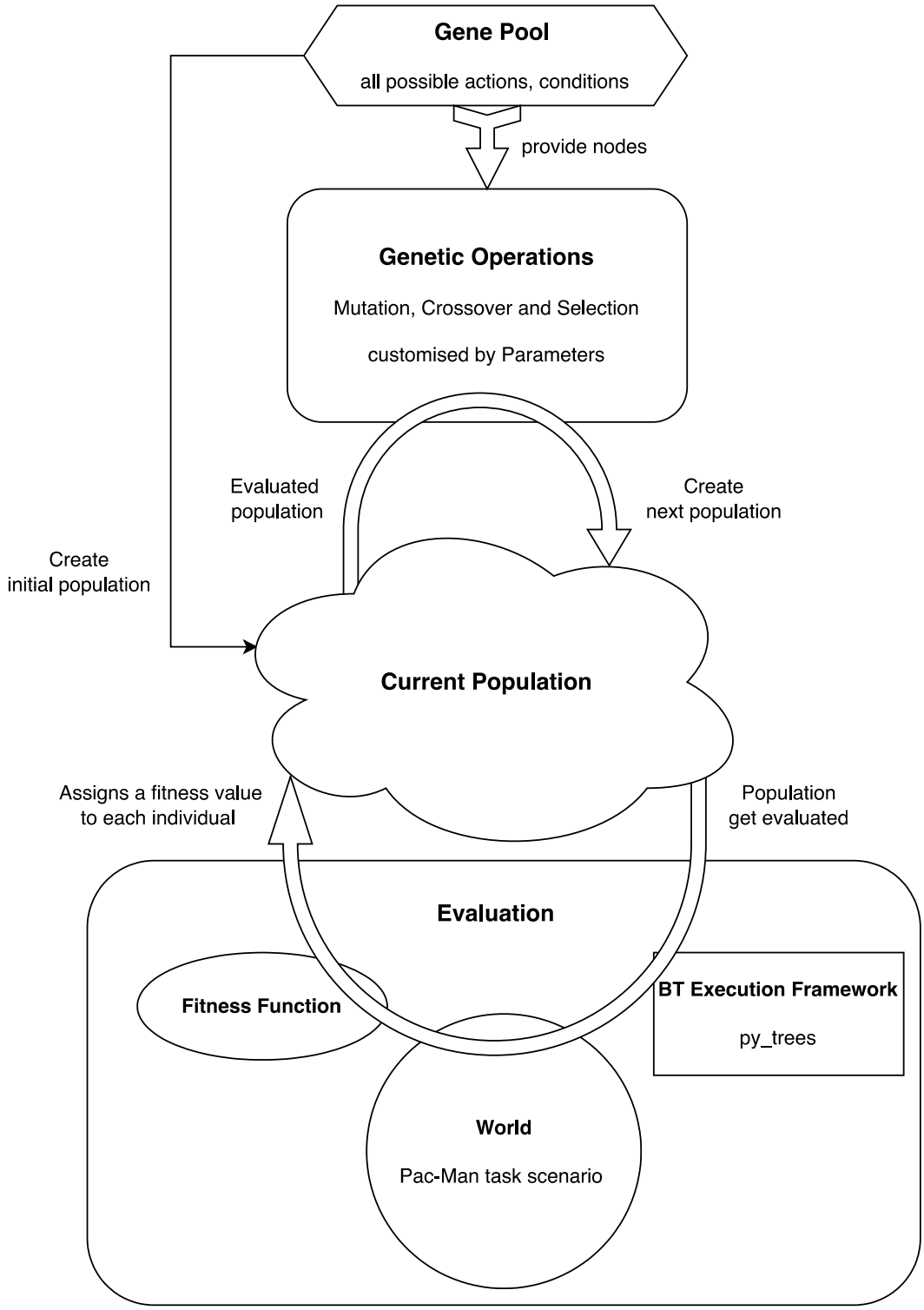


Figure 15: Schematic illustration of the main components in the Genetic Programming algorithm

Starting from the current population, the upper part represents the genetic operations and the gene pool. At the bottom, the evaluation process is visible, including the BT execution framework, the task scenario and the customised fitness function. The figure visualises the implementation based on the work of Iovino et al. (2021, pp. 4593–4594) and Escudero (2022), as well as our adaptions.

The main components of the Genetic Programming algorithm are visible in Figure 15. In the middle part, the current population is located, which is the central component, since all operations are done to evaluate and manipulate the contained BTs. In the lower part of the figure, the evaluation process is illustrated. As we mentioned, the framework “py_trees” executes the BTs in the task environment, after they are converted from lists of strings to a readable framework by the interface. After a BT passes one or several games in the Pac-Man scenario, the fitness function receives the score. In the case that several games per individual are conducted, which can be modified in the parameters, the arithmetic mean is used to calculate the score for the fitness function.

$$fitness = score + lifetime - costs - failure \quad (3)$$

Table 3: Factors of the fitness function in our Pac-Man benchmark scenario

α_{depth}	0.4
β_{size}	3.0
γ_{failed}	50.0
$\delta_{lifetime}$	0.2

We use equation (3) to calculate the fitness value. The used factors are summarised in Table 3. The costs are assembled from the multiplications of the factor α_{depth} with the depth and the factor β_{size} with the size of the BT of one individual in the population. Without this negative influence, the BTs grow unrestricted in size. If a BT returns a “failure”, the costs are added by another factor γ_{failed} multiplied with the number of received “failures” at the root of the respective BT. In this way, those BTs are preferred, which execute actions successfully or integrate reasonable conditions. A positive influence has the amount of ticks, which reflects the duration of an individual's survival. The summand “lifetime” in equation (*3*) is the result of the product of ticks multiplied by the positive factor $\delta_{lifetime}$. After this calculation, the final fitness value is assigned to each individual. If all individuals of one population are evaluated, the next generation evolves through

the genetic operations of mutation, crossover and selection. These operations can be customised in the parameters to receive optimal results. The final parameters and their descriptions for the Pac-Man example are listed in Appendix A.
Possible nodes for mutations are taken from the gene pool, which is visible in the upper part of Figure 15. All implemented actions and conditions in the gene pool are presented in Section 3.2.4. We like to mention that only the control flow nodes “sequence” and “selector” are available in the Genetic Programming algorithm.

3.2.4 Implemented Actions and Conditions

The architecture of the actions and conditions is inspired by the structure of the Pac-Man example presented by Colledanchise, and Ögren (2018, pp. 11–14). In addition to their approach, we added the two conditions “Pill Close” and “Big Pill Exist”, as well as the action “Approach Big Pill”. This becomes necessary since our benchmark task environment is more complex. All implemented execution nodes are listed in Table 4:

Table 4: Actions and conditions of our Pac-Man benchmark scenario

Actions:	**Conditions:**
Avoid Ghost	Ghost Close
Chase Ghost	Ghost Scared
Eat Pill	Pill Close
Approach Big Pill	Big Pill Exist

Following our design principles introduced in Section 2.1.5, the names of the execution nodes are meant to be self-explanatory in combination with a condition-action subtree structure. An exception is given by the action “Approach Big Pill”, which needs further explanation. As mentioned in Section 3.1.1, the agent Pac-Man has additional information about the position of the four big pills. This aspect is implemented in the mentioned action in the form of a list with the position of the power pills. If ticked (see Section 2.1.3), the program selects the primary action which minimises the distance between the relative position of Pac-Man and the big pill.

The actions are designed according to the design principles in sub-section 2.1.5. We tested each action and condition individually and within a tree structure to ensure their functionality. The action “Approach Big Pill” shows irregularities in its application. Due to the approach of shortening the distance between the agent and the big pill, in some areas on the game board, Pac-Man sticks in a loop between two nearby positions. Thus, the agent is not able to reach the remaining pills. We further address this aspect in Section 3.3.2.
We like to mention that it is not the intention to build an optimised BT for Pac-Man, but rather to test the implemented learning framework. In that sense, we did not optimise the actions down to the smallest detail due to limited time resources. By having a critical view from the designer's perspective, the action “Eat Pill” could be further optimised, so that an optimal path within the observation radius, concerning the most pills is planned constantly. In the current implementation, a list of primary actions is checked, beginning with “right”. Also the action “Approach Big Pill” could be further optimised, so that the agent does not get stuck in a loop. As a further optimisation, we suggest implementing an additional action “Search Small Pills”, which is executed if no more power pills exist.

3.3 Procedure and Observations

In this section, we describe the proceeding, the difficulties, and observations during the implementation and the learning process. In the first part, we present and explain the hand-made BT. We use the hand-made tree to test the execution nodes in their combination and to compare the hand-made BT with those learned by the Genetic Programming algorithm. In the second part, we present and describe the proceeding and the results of the Genetic Programming algorithm. We make a comparison in the third part.

3.3.1 Hand-Made Behaviour Tree

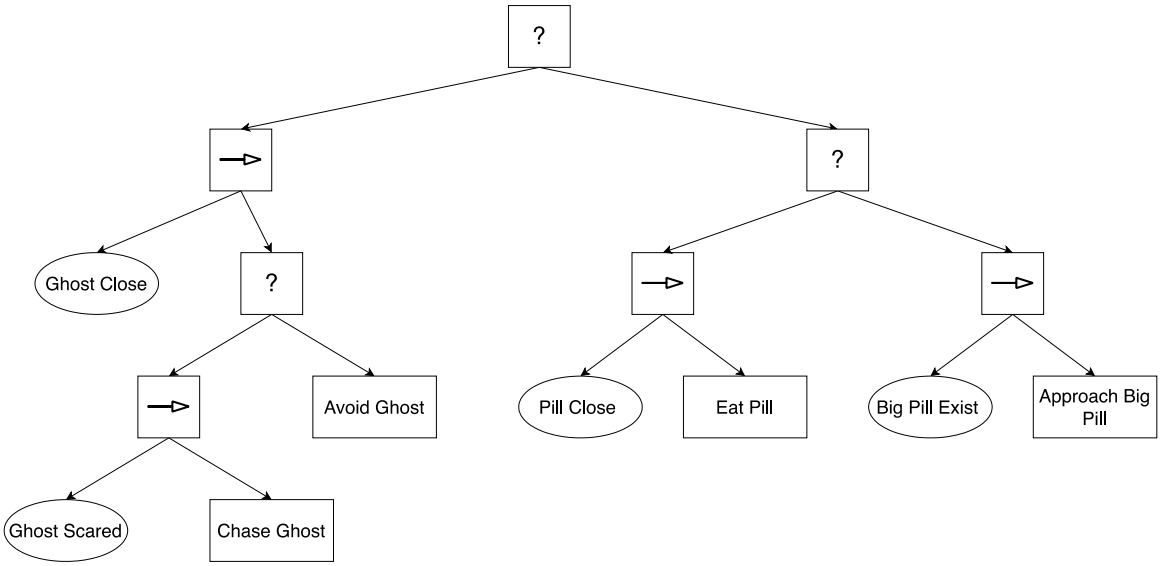


Figure 16: Hand-made BT for the benchmark example Pac-Man.

Adapted from Colledanchise and Ögren (2018, p. 13, fig. 1.12).

The hand-made BT is oriented on the implementation by Colledanchise and Ögren (2018, pp. 11–14). In the beginning, we aimed to reproduce their implemented tree within the setting described in section 3.2. The first BT looks like the hand-made BT, shown in Figure 16, but without the subtree on the right side. At the position of the selector node on the right side, only the action “Eat Pill” existed. In the implementation of Colledanchise and Ögren (2018, pp. 11–14), a greedy algorithm is used for this action. However, it is not mentioned in their approach, which area of the game board is observable by the agent. It can be assumed that the game board is fully-observable to the agent in their example. Since we limit the observability to a small circle in our scenario, as shown in Figure 12, we needed to implement the action differently. In that sense, Pac-Man approaches the closest pill in its observation circle. To test, if a small pill is close, we implement the corresponding condition.

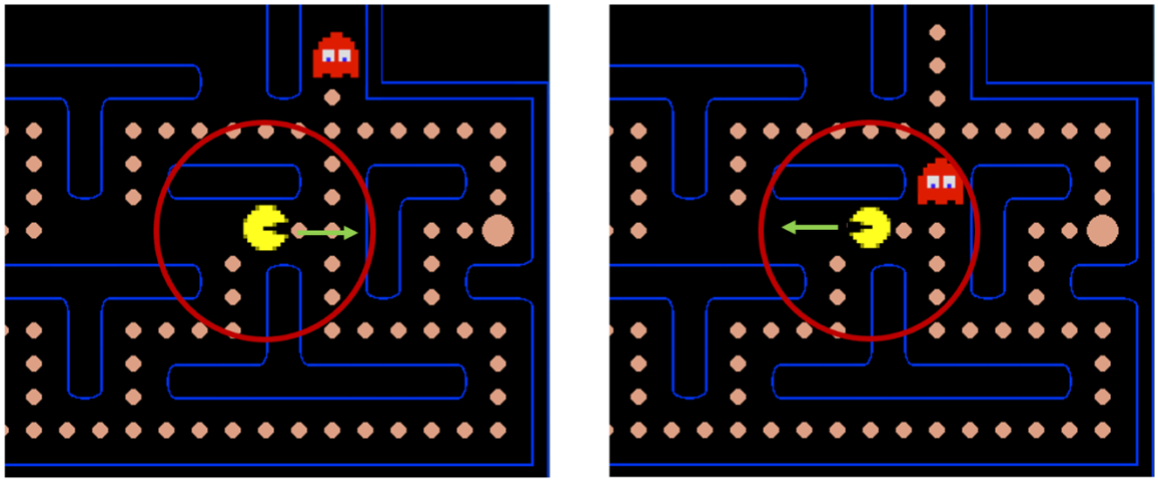


Figure 17: Illustration of the reactive behaviour in our Pac-Man benchmark example

Excerpt of the Pac-Man example with Pac-Man (yellow), observation circle (red circle), selected direction (green arrow) and one ghost (red ghost). The sequence shows the reactive behaviour of the agent. Left: The agent executes the action "Eat Pill". Right: The Ghost is inside the agent's observation circle and the action "Avoid Ghost" is executed.

At this stage, Pac-Man is able to collect pills, avoid ghosts and chase them, if it accidentally chomps a big pill. The agent shows reactive behaviour with regard to its enemies. An excerpt is shown in Figure 17. On the left side of the figure, Pac-Man is executing the action "Eat Pill" and selects the primary action "right". On the right side of the figure, the agent reacts due to its changed environment. By receiving an enemy inside its observation circle, the action "Avoid Ghost" is executed. The selected primary action is "left", with the goal of escaping from the ghost. Here, we observe a teleo-reactive behaviour, as we described in Section 2.2.1, since the agent reacts to its environment while it aims to collect pills.

However, if there are no pills inside Pac-Man's observation radius, the agent is not moving until a ghost approaches. To keep the agent moving, the action "Approach Big Pill" is implemented. Because the game board is not fully observable, the agent does not have the possibility to plan a path through the maze. As described, the action selects the primary action, which minimizes the relative distance between the agent and the power pill[7].

[7] Through the implementation of the action "Approach Big Pill", the agent turns into a model-based, utility-based agent. We summarized the properties of such an agent design in Section 3.1.3. We refer the interested reader to Russell and Norvig (2021, pp. 54–55). This is because the agent calculates the relative distances between itself and the big pill for all possible primary actions. Through that it predicts, how the environment would look like if it would perform one of those actions, to then select the best option. The selection of an action, which is based on the comparison of the calculated distances indicates a utility-based agent design.

The complete hand-made BT is shown in Figure 16. As mentioned, the left side handles the interaction with the ghosts. If no ghost is close, the agent can care about chomping pills. This right part of the BT is ticked, if the condition “Ghost Close” returns a “failure”. Depending on, if a pill is in the observation radius, the agent approaches this one or selects a primary action, which minimizes the distance to the next power pill. In this way, the agent prioritises to survive or to chase a ghost and otherwise chomp pills.

3.3.2 Learned Behaviour Trees by Genetic Programming

We followed an iterative process by implementing the Genetic Programming algorithm. In the beginning, we focused on eliminating smaller bugs in the algorithm, e.g. the synchronisation between the game tick and the BT tick. To calibrate suitable parameters, we adapted the configuration according to the resulting BT of every iteration step, which included the fitness function. One *iteration step* includes 10 *trials*. Each of the trials tried to run 100, 200, 500 and 1000 *generations*. The last two stages of 500 and 1000 generations were only possible to run in the latest 6 iteration steps, due to adaptions at the simulation framework. Each individual runs five times in the Pac-Man environment to calculate an average for the score, which is passed to the fitness function and assigned to each of them. This became necessary since the score of one individual varied widely. One population contains 16 individuals. The seeding, which forms the first generation, is randomly created, according to the set parameters.
We executed the first simulations on a PC with the specifications listed in Appendix A. At this point, the average time for one Pac-Man game was about 23,4 seconds. Such a long simulation time did not allow us to have a maximum of generations of more than 300. The borders of this implementation of Pac-Man are already reached with a four-times faster speed than the original game. The bottleneck is the “py_game” framework, which needs to render the whole game (Andrews et al., 2023). We created a version of the Pac-Man game without the “py_game” framework. As a consequence, we could drop the simulation time per game to less than 2 seconds. Through that, the duration for one simulation of one trial was about 36 hours. Because the PC needed significantly more time than the simulation

server to execute the 10 trials in parallel, we switched to the server, which is specified in Appendix B.

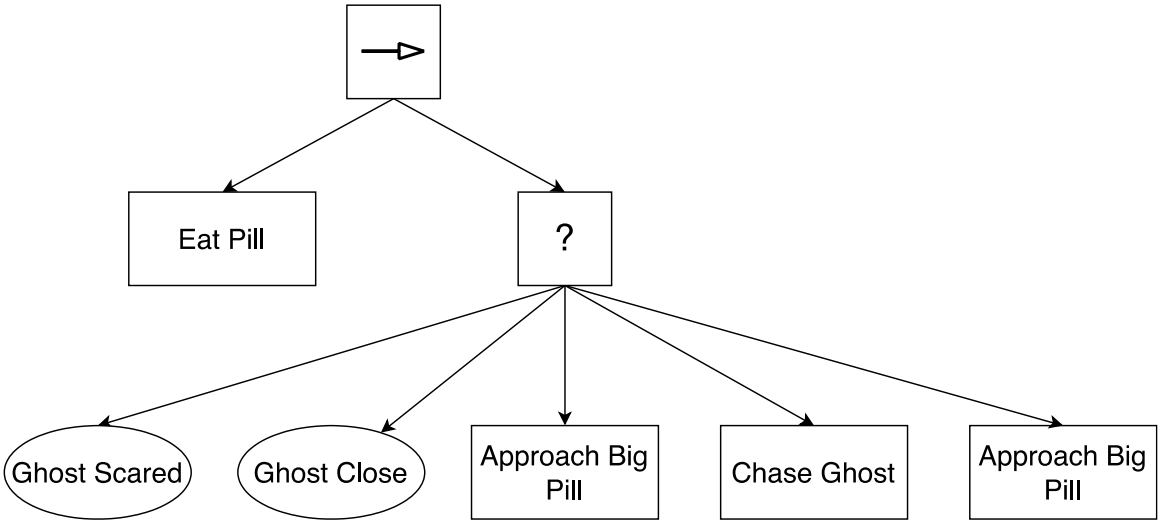


Figure 18: Learned BT from the latest trial of the Genetic Programming algorithm after 1000 generations.

The configurated parameters are listed in Appendix A. The BT has the property of stopping the loop of the action "Approach Big Pill", which is mentioned in Section 3.2.4.

After the mentioned adaptions above and 6 successful iteration cycles, we made an unexpected observation. One of the BTs learned by the Genetic Programming algorithm showed interesting behaviour about the action "Approach Big Pill". The BT is shown in Figure 18 and consists of two conditions and three different actions. The used parameters are listed in Appendix A. At first glance, it seems strange, because it is smaller than the hand-made BT and contains one action twice. We observed that it is able to stop the loop of the action "Approach Big Pill", which is mentioned in Section 3.2.4. This is done due to the structure of the selector node. If one of the two conditions returns a "success", the agent does not execute any action. Thus the loop is stopped. We like to mention that the learned BT does not contain any actions to protect Pac-Man from the Ghosts. We observed that the BT has a high fitness value on average and has survived through many generations. This can be explained through the design of the fitness functions, which reward the amount of survived ticks. The agent does not move after it leaves the loop, but it takes some time until the ghosts reach Pac-Man. In this way, the BT gets a high fitness value.

3.3.3 Comparison Between the Hand-Made and Learned Behaviour Trees

Table 5: Arithmetic mean, standard deviation, minimum and maximum scores of the hand-made BT and BTs learned by Genetic Programming for the Pac-man benchmark scenario

	Hand made	100 Gen	200 Gen	500 Gen	1000 Gen
Arith. Mean	1657,20	2063,46	2139,36	2085,67	2117,26
Standard Deviation	760,87	513,22	420,24	635,91	440,37
Min. Score	0	410	120	400	120
Max. Score	5070	4610	4620	5260	4620

To compare the hand-made BT with the ones learned by the Genetic Programming algorithm, we applied the BTs to the Pac-Man game again. We chose the respective best individual of the 10 trials of the latest iteration and calculated the arithmetic mean and the standard deviation over 1000 games. The results are presented in Table 5. The respective BTs are listed in Appendix C.

According to the Table 5, the individual with the highest average evolved after 200 generations. Our observations in Table 5 indicate that an increase of generations does not linearly correlate with an increasing score of the resulting individuals.

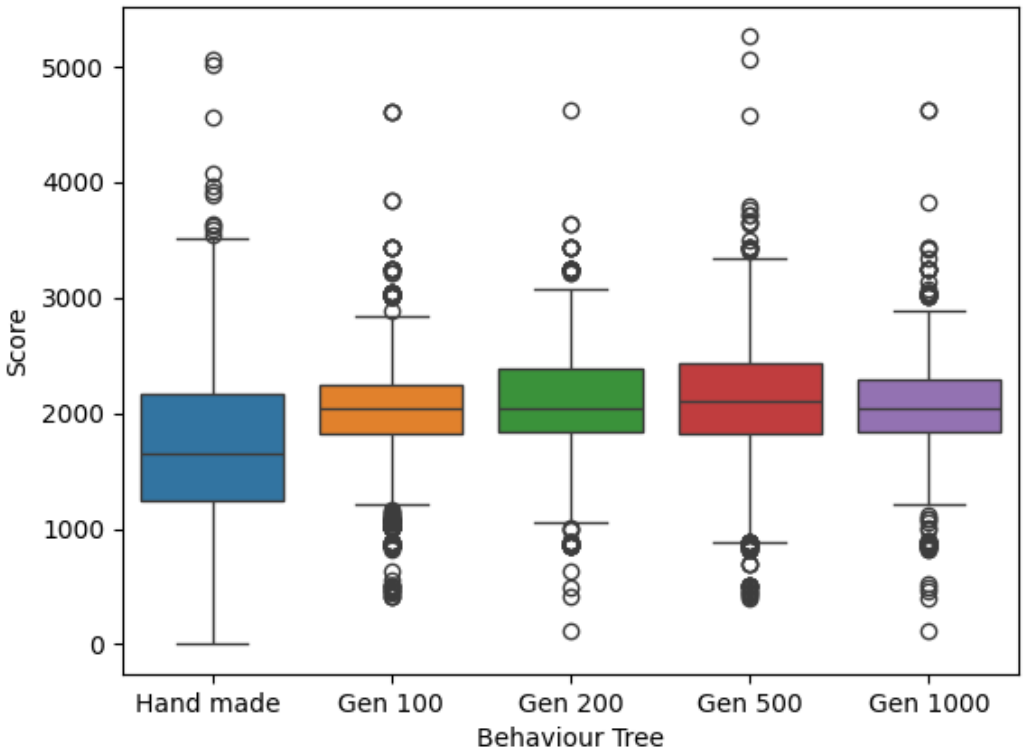


Figure 19: Boxplots of the score of different BTs.

The score is applied on the vertical axis. In the horizontal direction, the boxplots of the hand-made BT and the best BTs that evolved after 100, 200, 500 and 1000 generations are shown. Each boxplot shows the results of 1000 played games of one BT.

To visualise the data from Table 5, we create a boxplot, which is shown in Figure 19[8]. It is visible that the whiskers, as well as the first and third quartile of the hand-made BT, are wider than the others. This indicates that the learned BTs show a more robust behaviour in the game. A reason for that could be that the size of the learned trees is smaller compared to the hand-made BT and with that the variety in their behaviour.

In Table 5 and Figure 19, it is visible that the learned BTs have a similar arithmetic mean over 1000 games than the hand-made BT. We derive that the Genetic Programming algorithm found practical solutions to organise the different predefined nodes of the BT with regard to the score. However, we observed that some of the learned BTs, like the one in Figure 18, omit nodes to protect Pac-Man

[8] Do not be confused with the horizontal line in the box of the boxplots. According to their definition, this is the median and not the arithmetic mean, which is shown in Table 5.

from the ghosts. If we assume that the protection of the agent Pac-Man is a security aspect, the Genetic Learning algorithm fails to ensure this aspect in our benchmark scenario.

3.3.4 Lessons Learned for the Implementation of the Forklift Truck

The application of the Genetic Programming algorithm to the Pac-Man example shows several aspects, which can be taken into account for the implementation of the forklift truck “Crayler”. Since the proceeding for the Pac-Man example could be categorised as successful, a similar approach can applied to the forklift truck. In this way, the environmental analysis, suggested by Russell and Norvig (2021, pp. 43–47) provides a suitable tool to get an estimation of the necessary complexity of the agent’s design. During the process of development for the Pac-Man example, the agent design needed to be modified. For the action “Approach Big Pill” the distance for different action-caused new states are calculated and compared. Through that, the structure of the agent turns from a model-based agent into a model-based, utility-based agent, according to Russell and Norvig (2021, pp. 54–55). It can be said that a first environmental analysis can give a first indication of the needed structure of the agent, which can change during the process of further development.

As the case of the learned BT of Figure 18 shows, the resulting behaviour is not always predictable a priori. It was not intended that the loop of the action “Approach Big Pill” get stopped, even if it was a construction fault. Nevertheless, this unintended construction fault opens up the door for the described observation. This shows on the one side that even if the actions and conditions are strictly defined, their combination through the Genetic Programming algorithm can produce unpredicted, and possibly unintended behaviour. On the other side, the learning algorithm found a way to stop the loop, which was not found by us.

The Genetic Programming algorithm optimises the BTs according to the underlying fitness function. Because of this, the fitness function must have a special importance during the development process. It needs describe the task precisely so that the learning algorithm does not optimise past the task. Connected to this, it

is crucial to adjust the Genetic Programming parameters suitable after the fitness function is well-defined.

As shown in Section 3.3.3, the learned BTs reach a similar score than the hand-made BT. However, the automated creation of the BTs has the costs of investing more resources, particularly time and computer power, compared to the hand-made version.

The number of generations until the algorithm stops, is not proportional to a higher score, as visible in Figure 19. It seems that the amount of trials, and with that, different seedings, has more influence on the result than the number of generations. By comparing these observations with the literature, we found that authors appear to follow different strategies, independent of the amount of trials. Iovino et al. (2021) use rather small population sizes, but many generations, while Zhang et al. (2018) do it the other way around. Poli et al. (2008, pp. 26–27) recommend using a population size of 500 and a lower number of generations but also describe that small population sizes are used.

Another lesson learned is that aspects of security should not be implemented as separate actions and conditions. To guarantee a “safe” behaviour, caused by the learned BTs aspects of security should be included in actions with a coarser granularity. In this way, the nodes can be combined by the learning algorithm, without violating the security aspects. A negative example is provided by the learned BT in Figure 18.

Finally, a focus should be on simulation time. During the process of the implementation, it became necessary to revise the Pac-Man game and to use a more powerful computer to reduce the time per game. The time per simulation run is a crucial factor, as also Iovino et al. (2021) state. In this sense, the simulation time sets a boundary to the amount of possible generations and the size of the population.

4 Implementation of an AI System for the Forklift Truck “Crayler”

With the insights and lessons learned from the benchmark example of Pac-Man, a BT for the forklift truck “Crayler” should be learned. To do so we modified the Genetic Programming algorithm for the Crayler scenario. The design of the actions and conditions follow the insights from Section 3.3.4. Therefore, the proceeding is similar to the one in the Pac-Man example. In the first step, we describe and analyse the environment and the agent. Based on this, we adapt the learning framework and implement the actions and conditions. At the end of the chapter, we present our observations, possible improvements and limitations.

4.1 The Crayler Scenario

The project “Hopper”, which stands for “Handling of man-made Objects using automated Positioning, Planning and Enhanced Reasoning methods” was partly conducted at the AIT in cooperation with other project partners (*HOPPER - AIT Austrian Institute Of Technology*, 2022). The goal of the project was to automate subtasks of utility outdoor machinery. One of the developed vehicles within this project is a forklift truck, which is used to load and unload pallets. In this context, a scenario was created in which the forklift truck should collect pallets in a defined area and unload them at desired and predefined positions that are called *slots*. Since the subtasks are automated, the next step is to automate the execution of the single subtask towards an autonomous forklift truck.

Parts of that scenario are similar to the Pac-Man example. The collection of the big pills can be seen as an approach to the pallets in the forklift truck setting. But other aspects are different. The environment of the forklift truck is not a discrete maze, but rather an open space. There are also no enemies, which try to catch the agent. For the scenario of the forklift truck, security aspects need to be focused.

4.1.1 The Properties of the Agent

The agent in this scenario is a forklift truck. In addition to the skeleton produced by the company “Palfinger”, sensors, two computer units and control lights are added by a team of researchers at the AIT. This version of the forklift truck is commercially available on the free market and is used in many application areas, like logistics or technical emergency services (Palfinger AG, 2022). The designation in the product catalogue is “Palfinger Crayler BM 214”. It can be controlled using a remote control, but does not carry out any automatic actions. Based on this remote controlled forklift truck, a team of several researchers at the AIT automated several parts of the forklift truck within the project “Hopper”[9] (FFG - Die Österreichische Forschungsförderungsgesellschaft, 2019; *HOPPER - AIT Austrian Institute Of Technology*, 2022). The team equipped the vehicle with different sensors and additional actuators so that it was possible to automate sub-processes of the loading procedure. This included path-planning algorithms, as well as visual identification software, which is partly based on deep learning for visual object recognition, e.g. identifying pallets, and keeping track of the current environment for safety reasons. The approaching, lifting, and unloading are automated as well. We adapt the used term for this modified version of the forklift truck and call it *Crayler* or simply forklift truck in the following.

9

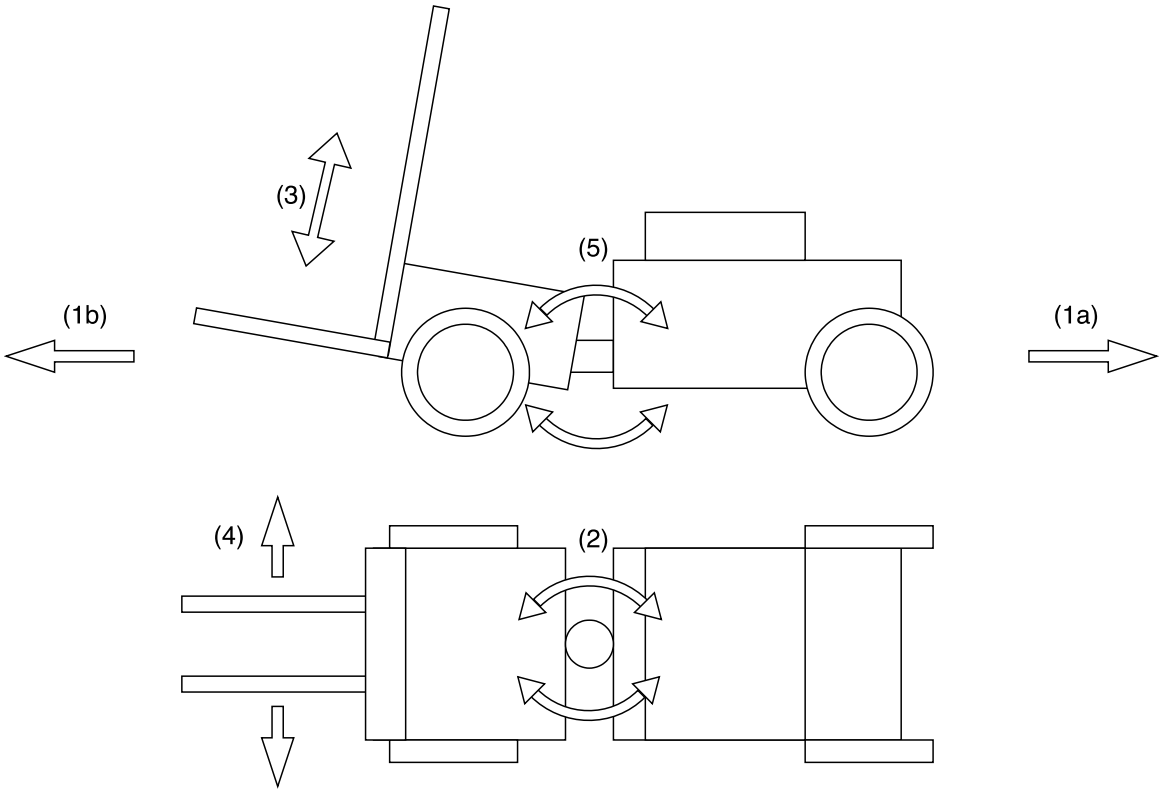


Figure 20: Schematic visualisation of the possible movements of the forklift truck

The *primary actions* of the forklift truck result from the direct execution of the actuators and are visible in Figure 20. These are:

- Driving forward (1a) and backwards (1b)
- Steering by moving the joint in the middle of the vehicle (2)
- Lifting and lowering the fork (3)
- Changing the horizontal position of the whole fork (4)
- Manipulating the angle of the vertical construction of the fork by changing the joint in the middle of the vehicle (5)

The exact actuation control is customised by the researchers of the AIT. In general, we assume that the control of the actuators is designed reliably, according to the designated tasks, except for one case. During the testing of the simulation, we observed that a pallet slid from the fork of the forklift truck, which was caused by incorrect loading. Due to this observation, it is not guaranteed that a primary action is executed correctly.

Figure 21: The modified forklift truck Crayler by the team of the AIT.

In Figure 21 the modified forklift truck is visible. In the figure, it is visible that the forklift truck consists of two parts, which are connected through a joint in the middle, which is manipulatable around a vertical and a horizontal axis. The so-called *backside* of the vehicle, the *fork* is located. The opposite side is the *forward* direction. With these components, the following listed *automated secondary actions* can be executed:

- *Plan path* from current position A to position B
- *Approach a pallet*
- *Load a pallet*
- *Unload a pallet* in a predefined position

The forklift truck can be remotely controlled by a person manually or by an external computer. With this system of the Crayler, controlling the vehicle with a BT is possible. For safety reasons, the forklift truck is equipped with a control light that is visible to a human if the vehicle is switched on. It is distinguishable between a

driving remote-controlled mode (yellow light), a loading remote-controlled mode (blue light), in which only the fork moves, and a fully automated mode (switching between blue, red, and yellow light).

4.1.2 The Properties of the Task Environment

In the real world, situations occur, in which empty pallets are randomly located in a defined area. Since the forklift truck is made for outdoor usage, we agreed with the development team at the AIT on a scenario, in which several empty pallets lying around at a construction site that are not needed anymore. The resulting task for the forklift truck is to collect all the unneeded pallets and place them at a defined location.

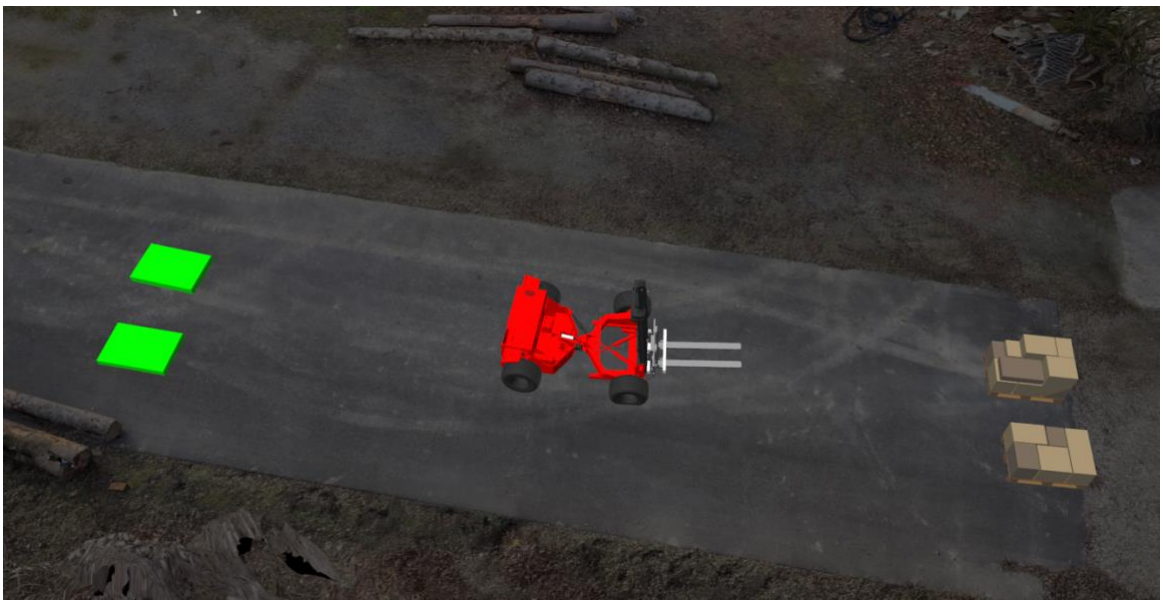


Figure 22: The virtual environment of the Crayler scenario

On the right-hand side of the screenshot, two unneeded pallets are visible. The green cuboids on the left-hand side are the defined slots to unload the pallets. In between those two, the red forklift truck is located. Within the testing site tree trunks are lying around and the ground is only partly asphalted so that the virtual environment models the real world accurately.

From this real-world scenario, a defined task scenario can be derived, which was developed in cooperation with the team at the AIT. The concrete task is to collect all desired pallets in the defined area and place them in predefined *slots*. Therefore, the forklift truck receives a list of the unneeded pallets and their desired slots to

place them. On the company premises exists a defined outdoor testing area for the forklift truck. This testing site was scanned by a 3D scanner so that it could be used virtually in a *physical simulation*. The virtual environment with the scanned testing site is visible in Figure 22.

We agreed on using two task scenarios that differ in their degree of complexity through the number of pallets. In the first step, we adapt the task scenario in Figure 22, so that only one pallet and one (green) defined placing position are placed in the virtual testing area. By using this task environment, the Crayler system should learn the *action sequence* for loading and unloading one pallet. This setting is scaled to two pallets and two defined placing areas, as shown in Figure 22, in the second step to demonstrate the scalability and test the ability to learn a *decorator* with the Crayler system.

4.1.3 Complexity Analysis of the Task Environment

Similar to the Pac-Man benchmark example, we conduct an analysis of the task environment, based on Russell and Norvig (2021, pp. 45–47). We use the results of this analysis to select an appropriate agent design. The BT is learned in connection with the virtual environment, as shown in Figure 22. Because of that, the analysis is applied to the simulated virtual environment. It is clear that the simulation is an abstraction of the real world and models a defined scenario with a smaller range of details. Also, other factors are omitted, like other agents or weather conditions.

In the case of the simulation, the task environment consists of the scanned test site, the one or two pallets and the predefined areas to unload the pallets. In this setting, the forklift truck has the task of collecting the pallets and placing them in the defined area by following a given sequence of the pallets and the defined areas. The forklift truck is able to observe a section of its environment by its cameras to observe its surroundings and identify pallets. Besides that, the agent has the relative location of the pallets and slots to itself. Another visual sensor system of the Crayler is able to measure the distance between the vehicle and a pallet in front of the fork. Since the task of the agent is to collect pallets and unload them in

defined areas, the performance measure counts the successfully loaded and correctly unloaded pallets.[10]

Similar to the previous Pac-Man benchmark example we categorise the task environment according to Russell and Norvig (2021, pp. 43–47). A short description of the six categories is given in Section 3.1.3 in the respective first paragraph of each one:

- Observability:

  From the three alternatives fully observable, partially observable and unobservable, we categorise the task environment of the forklift truck in the simulation as partially observable. As stated in Section 3.1.3, the criterion for the task environment to be partially observable is that not all (but at least some) relevant aspects of the performance measure are observable by the agent. Applied to the simulation of the Crayler, the forklift truck can observe only a section of its environment by its visual sensors. Task-relevant aspects, like a pallet could be out of this visible section. Because of that, we conclude that the task environment is partially observable in the case of the simulation.

  In a real-world application, more factors need to be taken into account. In such scenarios, pallets can be hidden behind obstacles or a hilly landscape can shorten the observability. Other agents could also be between the vehicle and the desired pallet so that there is no direct visual contact. By taking these factors into account, the task environment is partially observable in a real-world scenario, so the assumption of the virtual simulation holds for the real world.

- Number of Agents:

  In the case of the simulation, the forklift truck is the only agent in the virtual environment. Because of that, it can be classified as a single-agent task environment, according to the definitions given by Russell and Norvig (2021, pp. 44–45).

[10] We give a detailed description of the fitness function for the Genetic Learning algorithm, in which the performance measure is inherently expressed, in Section 4.2.3.

In contrast to that the task environment in a real-world scenario can be classified as multi-agent. Here, situations occur, where several agents are acting simultaneously. In the case of a construction site, it is thinkable that humans are involved. Also, other autonomous vehicles could cross the way of the forklift truck. The behaviour of all these agents can be seen as maximisation of their performance measure, which has an influence on the performance measure of the forklift truck, e.g. crossing its way. Because of that, they can be categorised as agents, which relates to the performance measure of the forklift truck (Russell & Norvig, 2021, pp. 44–45).

- Determinacy:
  In contrast to the Pac-Man example, the task environment of Crayler simulation is nondeterministic. As we described in section 4.1.1, we observed the case that a pallet slid from the fork. From the agent's perspective, this represents unpredictable interference.
  The real-world situation is also a nondeterministic task environment. Many other factors can influence the next state of the forklift truck’s environment, e.g. weather conditions. As described by Russell and Norvig (2021, p. 45), the agent is not capable of recognising all the possible influences in its environment. This also includes unobservable factors. In consequence, the real-world situation is too complex, so we conclude that the task environment must be seen as nondeterministic.

- Episodic or Sequential:
  The task environment is sequential because the forklift truck is moving within the defined area. Since the new position of the forklift truck affects the next decision, atomic episodes can be discarded, according to Russell and Norvig (2021, p. 45). The same counts for the application of a real-world setting.

- Static or Dynamic:
  In the case of the simulation, the task scenario is dynamic. As we described in section 4.1.1, we observed that a pallet slid from the fork of the forklift truck. In that case, the simulated physical system causes the movement, through which the environment changes without any interaction by the vehicle. It is thinkable that the forklift truck plans a suitable path while the environment is changing. Following the definition by Russell and Norvig (2021, pp. 45–46), the task environment is dynamic. The real-world environment can be treated similarly.

- Discrete or Continuous:
  Both, the simulated and the real-world task environment are continuous. This can be derived from the aspect, of how time is handled (Russell & Norvig, 2021, p. 46). If the forklift truck is in movement, the environment appeals to the agent as a continuous change. Although this change is sensed with a sampling rate to be processable by a computer, it is globally seen as a continuous task environment in both cases.

- Known or Unknown:
  As mentioned in Section 3.1.3, this category refers more to the agent and its designers than to the environment. In that sense, the decisive point is the knowledge of the agent and designer about the environmental “laws of physics” (Russell & Norvig, 2021, p. 46). In the Pac-Man example, the rules of the game are comparatively simple. Applied to the simulation of the forklift truck, this is not the case. The sliding of the pallet from the fork, as we described in section 4.1.1, is a disruptive and unpredictable factor from the perspective of the agent. From this case, we conclude that the task environment is unknown. The same count for the application in the real world.

Table 6: Categorisation of the task environments

Pac-Man Environment	Simulation	Real World
partially observable →	partially observable →	partially observable →
multi-agent ↗	single-agent ↘	multi-agent ↗
deterministic ↘	nondeterministic ↗	nondeterministic ↗
sequential ↗	sequential ↗	sequential ↗
dynamic ↗	dynamic ↗	dynamic ↗
discrete ↘	continuous ↗	continuous ↗
known ↘	unknow ↗	unknow ↗

In the first column starting from the left, the task environment of Pac-Man is analysed. This is followed by the simulated environment of the forklift truck in the middle and an imagined real-world scenario on the right side. Similar to Table 2, categorisations with a low complexity are marked in green and sign (↘), in between in yellow and sign (→), and with a high complexity in red and sign (↗). Adapted from Russell and Norvig (2021, p. 47, fig. 2.6).

Concluded the task environment of the simulation is partially observable, single-agent, nondeterministic, sequential, dynamic, continuous and unknown. It differs from the task environment of the Pac-Man benchmark scenario in four categories, which are visible in Table 6. The task environment of the simulation is single-agent, continuous, nondeterministic, and unknown instead of multi-agent, discrete, deterministic, and known in the case of the Pac-Man scenario. In sum, the task environment of the simulation shows a higher degree of complexity compared to the Pac-Man scenario. The task environment of the forklift truck in a real-world scenario is assessed as even more complex than the other two analyses. According to Russell and Norvig (2021, p. 46), it is the hardest case to design an agent, except for the category of being observable, which we accessed as “partially observable”.

Based on the analysis of the task environment, it is possible to select a suitable agent program. A goal-based, utility-based agent design seems suitable for the forklift truck because the agent program keeps track of the environment and selects actions according to its utility function (Russell & Norvig, 2021, pp. 51–54). By having a look at the available actions in Section 4.1.1, this assumption holds. E.g.

the action “Plan path from current position A to position B” requires a model of the environment, which tracks obstacles and chooses the shortest possible path. It needs to be mentioned that these parts of the agent's program are implemented in deeper structures than the level, which is used by the learning algorithm. To be concrete, the learning algorithm uses the action, e.g. “Plan path from current position A to position B” as a whole and does not learn its subparts. In this sense, the aspects of model and utility are partly inherent within the used nodes, similar to the example of Pac-Man.

4.2 Implementation Framework

We adapted the majority of the software from the Pac-Man implementation to the Crayler scenario. By following the initial idea to use the tested BT learning architecture for the Crayler scenario, the main task in this implementation consists of connecting the learning framework with the virtual environment, which was provided by the researchers of the AIT. Besides that, several adjustments needed to be configured, like the fitness function.

4.2.1 Used Parts from the Pac-Man Example

As mentioned, the majority of the BT learning framework, which we tested in the Pac-Man scenario can be used for the application of the forklift truck. By recalling Figure 14, which shows the three-layered structure of the program, the second and the third layer can be taken over. These are namely the BT framework, which uses the software “py trees” (Stonier, 2023) and the learning algorithm based on the repository “Learning Behavior Trees using Genetic Programming” (Escudero, 2022). From the perspective of the layered structure, the first layer, which includes the scenario needs to be replaced.

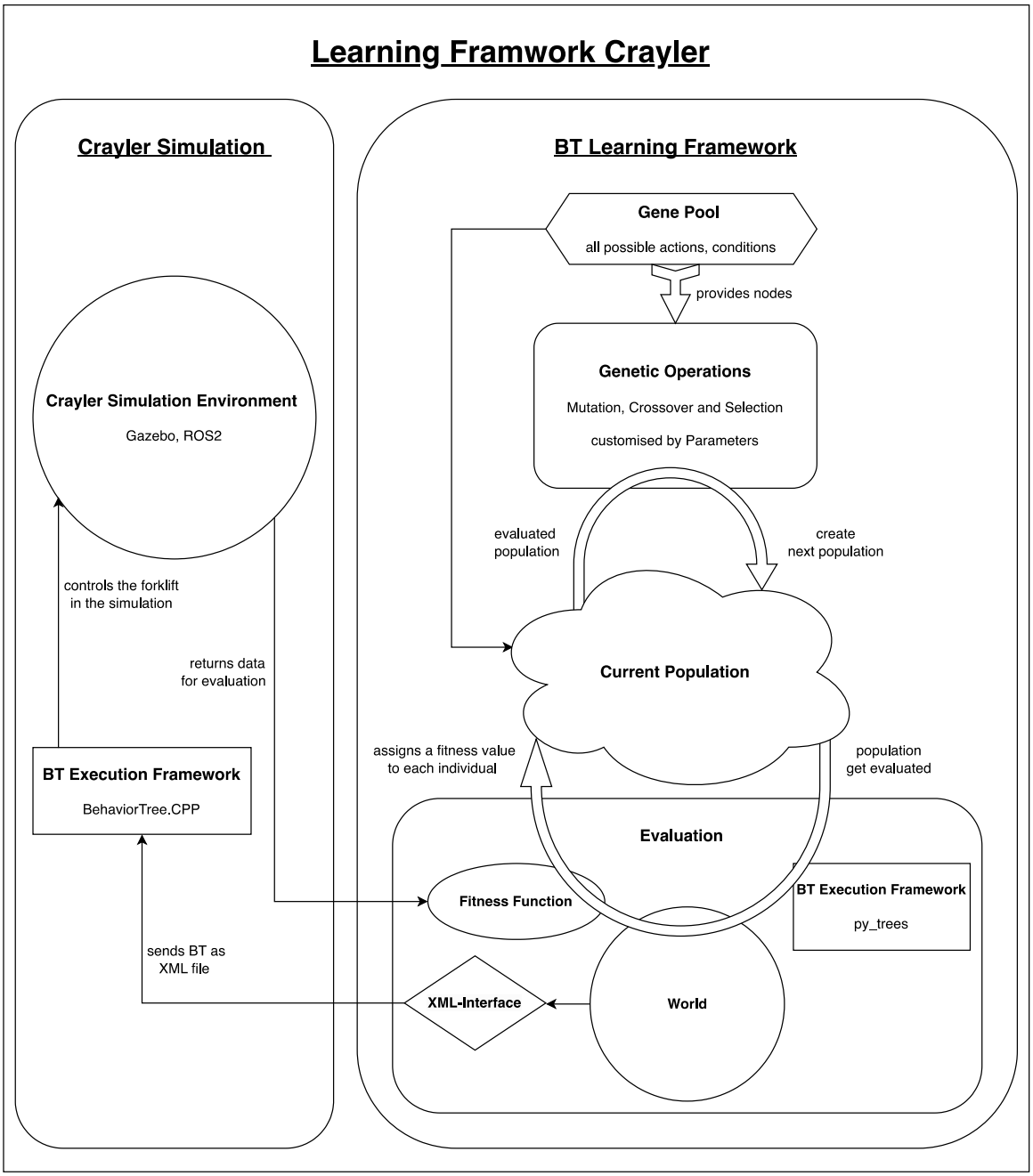


Figure 23: The Learning Framework of the Crayler

On the right-hand side, the BT learning framework, which was developed within the Pac-Man example is shown. The program is adapted in such a way that it is compatible with the Crayler simulation that is visible on the left-hand side. Therefore, an interface converts the BTs into XML files, which are readable by the Crayler simulation. The simulation, which is provided by the AIT returns data to the fitness function. The figure visualises the implementation based on the work of Iovino et al. (2021, pp. 4593–4594) and Escudero (2022), as well as our adaptions.

The learning framework for the “Crayler” is shown in Figure 23. On the right-hand side, the tested BT learning framework is visible, which includes the second and

third layers. Within the learning framework, it was necessary to adapt two parts. The first one is the program called “World”. This program has the task of providing a scenario in which the different BTs can be evaluated, while the BT execution framework controls the agent. In the case of the Pac-Man example, it was the Pac-Man game. For the forklift truck implementation, this part of the program needs to be externalised, so that the Crayler simulation could be included. The second part of the learning framework, which needed to be adapted is the fitness function. An exact description is given in Section 4.2.3.

4.2.2 Crayler Simulation and XML Interface

On the left-hand side of the figure, the Crayler simulation is visible. This part serves as the scenario for the learning framework and was provided by the researchers of the AIT. The software package of the simulation uses a different framework to execute BTs, which is written in the programming language C++ (Iovino et al., 2022). Because of that, it was necessary to develop an interface. The BT executor can read a specific XML format. Since the BTs are represented in lists as strings within the learning framework, they need to be converted. For this purpose, we developed the XML interface. The interface converts the list of strings into one XML file, which can be read by the BT executor of the simulation. Within the interface, the secondary actions are assembled into primary actions, necessary conditions are checked and a Blackboard is added and manipulated to keep track of internal and external states. These steps are necessary to connect to the existing simulation framework of the forklift truck.

4.2.3 Used Fitness Function

By designing our *fitness function* we aim to define those regions in a *search space* that solve our problem (Poli et al., 2008, p. 24). In the case of the Crayler scenario, the forklift truck should transport pallets form place A to B correctly. As another constrain, we force BTs, which are as small as possible according to their size and depth. According to Poli et al. (2008, pp. 75–76) a linear combination can be used to combine these different objectives in one fitness function.

$$f(t,l,c,d,s,x,o) = \alpha_{trans}\ t + \alpha_{loaded}\ l + \alpha_{compl}\ c$$
$$-(\gamma_{depth}\ d + \gamma_{size}\ s + \gamma_{failed}\ x + \gamma_{timeout}\ o) \quad (4)$$

Score:

t … number of the transported pallets

l … Pallet correctly loaded

c … BT successfully completed

Costs:

d … Depth of the BT

s … Size of the BT

x … BT failed

o … Timeout

Table 7: Factors for the fitness function in our Crayler scenario

Score:		**Costs:**	
α_{trans}	1000.0	γ_{depth}	0.3
α_{loaded}	8000.0	γ_{size}	4.0
α_{compl}	100.0	γ_{failed}	50.0
		$\gamma_{timeout}$	10.0

To assign a scalar *fitness value* to each individuum of the population, we used *the* fitness function in equation (4). We name the first three products of the fitness function “score”, because they increase the fitness value and represent the reward. The last four products decrease the fitness value and are assigned as “costs”. Each of the seven variables has its own factor, which is indicated by α for the score and by γ for the costs. The factors are listed in Table 7.

To calculate the fitness value, the Genetic Learning algorithm receives additional data from the simulation that are incorporated through the variables t, l,o, and x.

The variable $t \in \{0,1\}$ checks the correct transportation of a pallet. That means, the counter is set to 1 if a loaded pallet has moved from position A to the desired slot B. This does not include that the pallet is correctly lifted and unloaded by the forklift truck. The variable $l \in \mathbb{N}$ is increased by one if a pallet is loaded and unloaded successfully. In the case that l is increased, the variable t is set to 0, to indicate that the transportation is completed. The combination of these two counters ensures that the pallet is not pushed by the forklift truck, instead of correctly loaded. Additionally, the variable $c \in \{0,1\}$ checks, if the root of the BT receives a “success”, which indicates that the BT runs successfully.
The cost are calculated by the variable $d \in \mathbb{N}$ that takes the *depth* of the BT into account and the variable $s \in \mathbb{N}$ that increases with the *size* of the BT. In addition to them, the variable $x \in \{0,1\}$ ensures that the fitness value of BTs is decreased if the root receives a “failure”. To ensured that the simulation does not run endlessly with aimless combinations of nodes, the variable $o \in \{0,1\}$ decreases the fitness value of a BT if the simulation runs into a timeout, which we describe in more detail in sections 4.3.1 and 4.3.2.

4.2.4 Implemented Actions, Condition and Decorator

As we mentioned in Section 4.1.1, the available actions are combinations of primary actions added with additional software. The action “Plan path from current position A to position B” provides an example. To execute this automated action, several components are involved, like the control of the actuators for the primary action “driving forward”, visual sensing and several other software components. We chose actions deliberately with coarser granularity. This follows from the lessons learned in Section 3.3.4. To recap, it we mentioned that security aspects should be included within a learnable action so that these do not get violated. In the case of the action named above, it must be ensured that a safe path is selected, on which the vehicle does not hit any obstacles, loses its cargo or touches any person. Especially in real-world environments, in which humans are involved, the vehicle must react dynamically to changes in its surroundings. Such a reactive behaviour can be implemented within a BT. The secondary actions involve combinations of the primary actions that are implemented as human-designed,

hand-made BTs by the researchers of the AIT in consultation[11]. Within these tested sub-BTs necessary conditions are checked, values on a Blackboard are set and primary actions are executed. As we mentioned in Section 4.2.2, the sub-BTs are assigned to the secondary actions within the XML interface.

Table 8: Secondary actions in our Crayler scenario

Actions:
Approach Pallet
Load Pallet
Approach Slot
Unload Pallet

The actions in the gene pool (see Figure 22) of the learning framework are derived from the secondary actions mentioned in Section 4.1.1. The available actions are listed in Table 8. Because of the security aspects, necessary conditions are not available in the gene pool. They are included in the different actions. An example is the action “Load Pallet”. The sub-BT in this action includes a condition that checks if the pallet is successfully placed on the fork. For sure, it would enhance the human readability of the BT, if e.g. this condition would be included in the learned BT, according to the design principles described in Section 2.1.5. But as mentioned in the Pac-Man example, it is necessary that the learnable actions are safe by themselves[12]. A pallet, which is not loaded correctly could fall down during e.g. the lifting and pose a safety risk. Other authors, who learned BTs with evolutionary algorithms also faced the aspects of safety guarantees. Iovino et al. (2021, p. 4592) learned BTs for a robotic arm and stated that modular BT structures are suitable for genetic programming, without losses to the safety guarantees.
As described in Section 4.1.2, we conduct the learning process in two steps. While in the first step, the loading of one pallet is focused, in the second step we like to

[11] Please note that these hand-made BTs for each of the secondary actions can fulfill security aspects. They need to be tested, but once they are approved, it is possible to learn this approved secondary action as a “safe” part of a bigger BT. Theoretically, these sub-BTs can also be learned by an algorithm, but a continuative testing of each learned BT to ensure security aspects would increase the simulation time significantly.
[12]The safety of a BT or a sub-BT can be mathematically proven by a *phase portrait*. The principle idea is to ensure that risky areas in the *state space* are avoided. The interested reader is referred to Colledanchise and Ögren (2018, pp. 63–81, chap. 5).

show that the learning framework is scalable to more pallets. This can be done by learning a decorator. We designed this decorator as a *repeater*. The representation is shown in Figure 8. Here the “decorator” is on top of another separated node, which is influenced by it. The used learning framework does not offer the possibility to learn decorators. Because of this, a new node is implemented. The new node, which is in the following called “repeater” has the *policy*, to repeat until its decorated node returns "Success” from all children. The parameter n can be manipulated and represents the number of attempts. In this scenario, the parameter is manually set to 2 attempts.
To check, if all pallets are successfully loaded, the condition “All Loaded” is implemented and added to the gene pool. Through this condition, it gets possible to learn the repeater, since it only returns “Success”, if all pallets are loaded.

4.3 Procedure and Observations

In this section, the learning procedure is described. At first, we show that the simulation time is crucial and needs special attention. In the following the proceeding of learning the BTs for different scenarios is described. Here, we show that BTs can be learned for the different tasks. The adjustment of a suitable granularity of the actions plays a major role. At the end of this section, we suggest possible improvements and further steps from a technical perspective.

4.3.1 Used Hardware and Simulation Time

All parts of the simulation and the learning algorithm are running on the mentioned server. Its specifications are listed in Appendix B. In contrast to the Pac-Man benchmark scenario, it is not possible to run several trials in parallel. This is due to the limitation that we cannot run several physical simulation in parallel. Therefore, only one trial with one running simulation could be processed.
Another restricting factor is the possible speed-up of the simulation. It turns out that this part of the learning framework is the bottleneck. Since we use a physical simulation, that causes expensive computational costs. In this respect, the simulation can be speed-up by a factor of 3, compared to real-time. The whole

loading process for one pallet, with the positioning used in the simulation, takes about 1 minute in real-time. In consequence, a successfully conducted simulation of one pallet takes about 20 seconds. This sets a boundary and limits the possibilities of learning BTs with a Genetic Programming algorithm concerning the number of generations and the size of the population. By using a population size of 16 individuals and 500 generations, similar to the Pac-Man benchmark, one trial takes about 44 hours.

4.3.2 Learned Behaviour Tree for Loading One Pallet

The goal of the first step of the learning process is to demonstrate that one assigned pallet can be autonomously transported to a desired slot. Here, we focus on learning a BT, in which no unpredictable behaviour occurs through unconsidered combinations of nodes, which occurred in the Pac-man example. The selection of the action is taken carefully concerning the appropriate granularity. The selection process of the actions and its adjustments in collaboration with colleges at the AIT needed several steps to find a suitable granularity[13]. On the one hand side, the actions should not be too coarse-grained, so that the whole loading process is described in one action. On the other hand side, a too-fine granularity of the actions requires more episodes for a successful learning process, since the size of the BT gets larger and more possible combinations of nodes are possible. A large number of episodes require in consequence more simulation runs. Since a successful run needs about 20 seconds, the number of episodes is limited, which has again an influence on the granularity of the actions. The result of this selection process are the stated actions in Section 4.2.4.
For the first step, the task environment, which is shown in Figure 22 needed to be adapted. We removed one of the pallets and one of the desired slots, leaving only a single pallet and desired slot in the simulation. Within this task environment, the different BTs get evaluated through the fitness function of equation (4).

[13] To state the distribution of this cooperative work here again, the simulation of the Crayler and the concrete implementation of the subtrees of the used actions is provided by researchers from the AIT. A sharp line to my work can be drawn, by looking at Figure 23. The researchers provide the “Crayler Simulation” on the left side, while on the right side, the BT Learning Framework is implemented by myself.

At the beginning of the learning process, several problems with the simulation occurred, which we could solve. To avoid that some BTs run in endless simulations forever, we set a timeout of 40 seconds in simulation time. This results from the 20 seconds that are needed for one successful run (see section 4.3.1) plus a generously selected buffer of 20 seconds, which was suggested by a researcher of the AIT, who developed the Crayler simulation. The buffer serves as a compromise that, on one hand, ensures a pallet can be loaded and unloaded, while on the other hand, it limits the simulation time for unusable BTs.

During the process, we adjusted the fitness function concerning the size of the BT and the score factors. Score factors ranging from 10^1 to 10^2 were too small to ensure that the task could be learned, as the cost factors had an equal influence on the fitness value due to their similar range. So we decided to focus on the reward to ensure that the task could be learned and adjusted the score factors to the range of 10^2 to 10^4.

We also adapted the parameters of the Genetic Programming algorithm, listed in Appendix D. Since the simulation time is crucial, we reduced the number of individuals, which are held in the population from 16, used in the Pac-Man example, to 8. Through that, we could reduce the number of episodes. As the resulting effect of this reduction, the variability within one population is smaller due to the reduced population size.[14] We increased the number of generations slowly by starting with 100 over 150, to finally 200 generations.

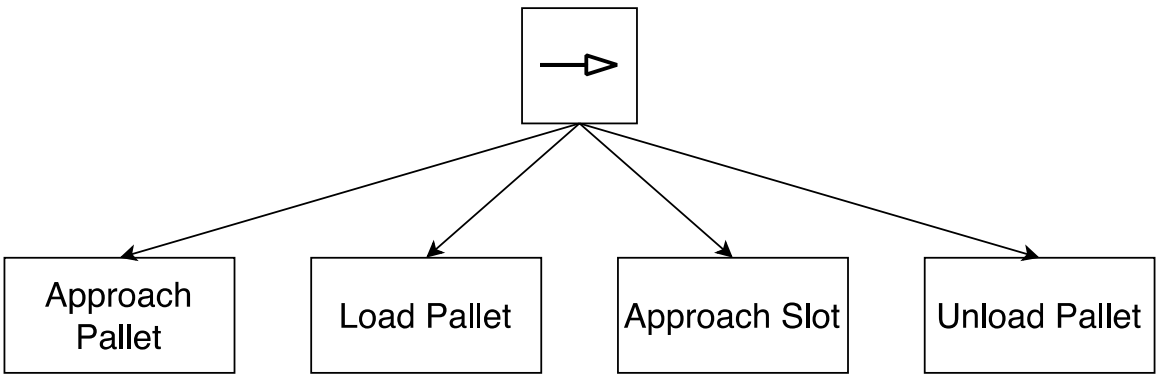


Figure 24: Learned BT for loading one pallet after 200 generations.

[14] Poli et al. (2008, pp. 26–27) note that the effect of *mutations* on the next generation increases with smaller population sizes while the effect of *crossovers* decrease.

The 83rd trial was successful and could manage the given task after 200 generations and a population size of 8. The whole trial took 9 hours and 9 minutes. The learned BT is shown in Figure 24. It consists of a sequence of actions. Other solutions with further branches are thinkable, but this solution has the smallest possible size to manage the given task. The result shows that the task scenario of loading and unloading one pallet is successfully learned by the agent.

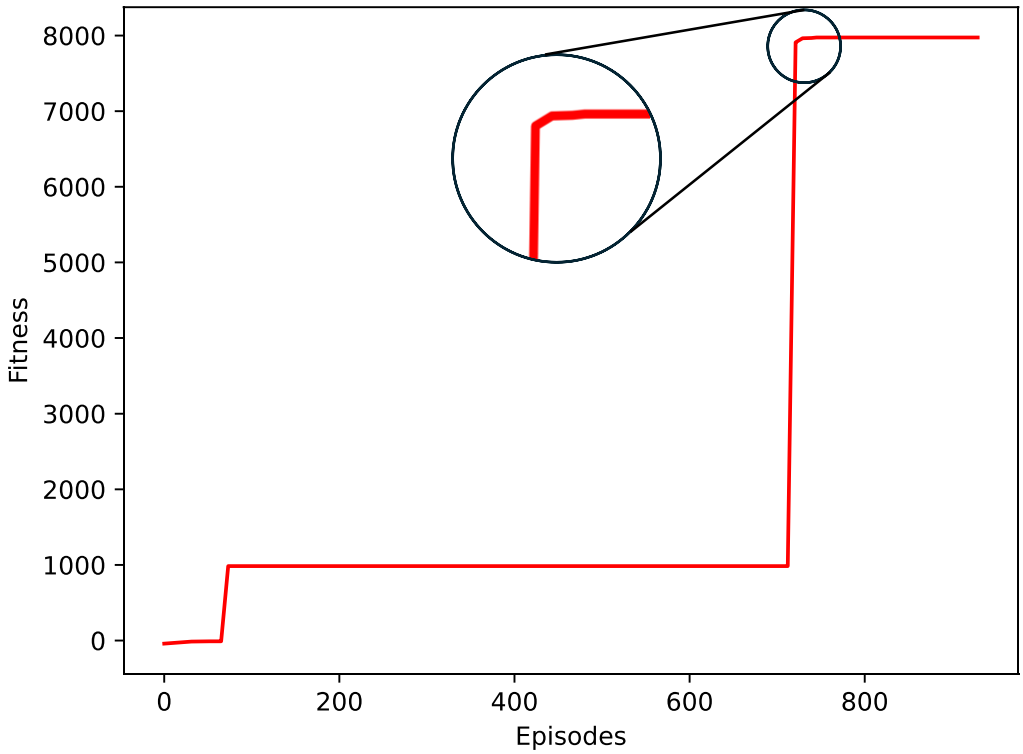


Figure 25: Learning curve of the 83rd trial for loading one pallet.

On the vertical axis, the highest fitness value of the best BT is plotted. The number of episodes is plotted on the horizontal axis.

By having a look at the learning curve of the 83rd trial in Figure 25, we can comprehend the learning process on the basis of the fitness value. On the horizontal axis, the number of episodes is plotted. The respective highest fitness value of the best individual is plotted on the vertical axis. We deduce that the first BT, which could transport a pallet evolved after about 80 episodes. At this point, the counter for the first score value in the fitness function increased. The next step happens after around 700 generations. Here, the pallet is successfully loaded and

unloaded at the desired slot. That increased the second score value of the fitness function and decreased, as described in Section 4.2.3, the first score value. It is visible that the optimum is not reached directly. In Figure 25, it is visible that the learning curve shows a tiny slope at around 720 episodes. This indicates that a BT evolved, with lower costs according to its size.

4.3.3 Learned Behaviour Tree for Loading Two Pallets

In the second step of the learning process, we aim to load two pallets successfully. To do so, the task environment is designed as shown in Figure 22. We added two nodes to the gene pool, which was used for the first learning step, as we mentioned in Section 4.2.4. These are a decorator and the condition “All Loaded”.

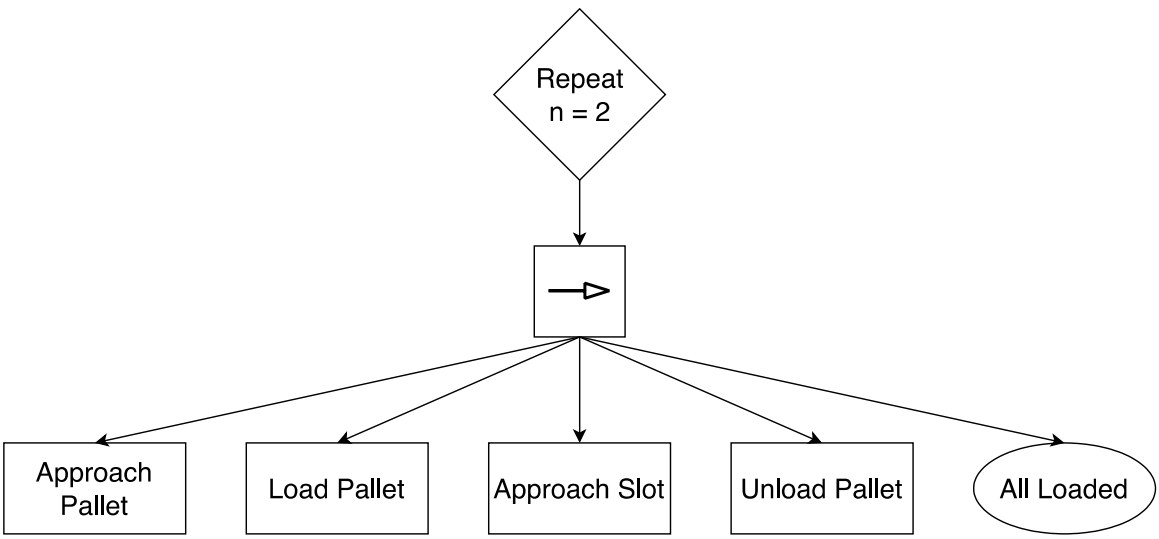


Figure 26: Learned BT after 400 generations for loading two pallets

In this BT a decorator, which repeats the sequence two times could be learned.

Increasing one pallet to two pallets requires an adjustment of the timeout (see section 4.3.2). We doubled the timeout of 40 seconds and tested it in the simulation. Because also the buffer of 20 seconds doubled, the forklift truck had enough time to move from the first slot position to the second pallet position. Due to a well-adjusted fitness function and suitable set parameters of the Genetic Programming algorithm (Appendix E), which can be deduced from the successful result of the previous step, 6 trials were necessary to learn the BT that is shown in

Figure 26. With each trial, we held the same or increased the number of generations. The last two attempts iterated for 400 generations, but only from the latter a BT evolved that could load both palettes. This is because of the different seeding, which is mentioned in Section 3.3.4. To run the whole last learning trial, it took about 33 hours and 38 minutes.

The learned BT in Figure 26 shows a sequence, which is repeated by its decorator. By following the policy, the sequence is repeated until all children return “success”. Since the condition “All Loaded” compares the already successfully loaded pallets with the desired number of loaded pallets, it only returns “success”, if the counter is set to two. By using the available nodes in the gene pool, the Genetic Programming algorithm generated an optimal BT concerning its size. If the BT is reduced by one of its nodes or differently assembled, it cannot fulfil the task.

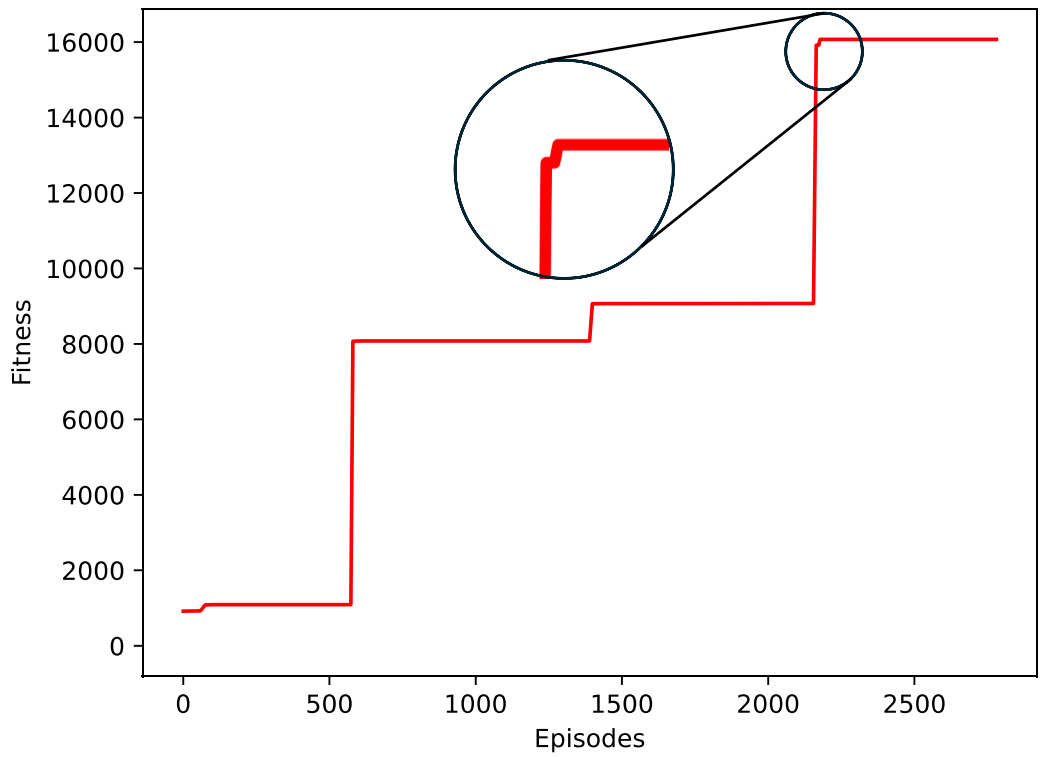


Figure 27: Learning curve for loading two pallets.

On the vertical axis, the highest fitness value of the best BT is plotted. The number of episodes is plotted on the horizontal axis.

This optimisation is visible in Figure 27. The graph is similarly constructed to the one shown in Figure 25. It is visible that the seeding was lucky. In contrast to the seeding of the first learning step, this initial population included one BT, which was able to move a pallet. We trace this back to the fact that the first BT has a fitness value of around 1000, which indicates that the variable t of the fitness function (see equation (4)) was set to one. Thus, it can be explained that the first trial with 400 generations did not succeed, because of an “unlucky” seeding.
From around 100 to around 600 episodes, a BT evolved that can correctly load and unload one pallet. At around 1400 episodes, we observe again that the variable t is increased, which indicates that the forklift truck can correctly load and unload one pallet and can move the second pallet. Both pallets are loaded and unloaded correctly at around 2100 episodes. The little slope that is visible at around 2100 episodes indicates an optimisation regarding the size.

4.3.4 Observations and Practical Experiences

The observations are summarised in the following, which we refer to in the next chapter. During the learning process, it turned out that some topics need special importance to achieve usable results. The environmental analysis at the beginning of the chapter in section 4.1.3, estimates the level of complexity. Although the Crayler scenario is similar to the one in the Pac-Man benchmark, it differs, concerning e.g. the need for safety guarantees within the actions, the complexity of the task environment and the number of nodes in the gene pool.
We observed the following aspects during the process of implementation and application of the Genetic Programming algorithm. In a wider sense, they can be understood as experiences and application instructions:

- The user of the Genetic Programming algorithm needs a rough idea of how the learned BT should look like. In other words, the user must ensure that the available nodes in the gene pool can be used to assemble a BT that offers a solution for the given task. Poli et al. (2008, p. 23) call this property of the gene pool “sufficiency”. By choosing the nodes in the gene pool the user defines the *search space* of the Genetic Programming algorithm (Poli

et al., 2008, p. 24). In the case of the Crayler scenario, these are all possible BTs that can be assembled from the given gene pool.

- The fitness function directs the Genetic Programming algorithm. Therefore the fitness needs to be measured as precisely as possible to define the task. Poli et al. (2008, p. 24) describe that the fitness function defines the areas in the search space that solve the given problem. We differentiated between the score and costs in our fitness function (see equation (4)) to achieve two different objectives, namely the correct loading and unloading of the pallets and an optimum regarding the size of the BTs. This approach of multiple objectives that are linearly combined in the fitness function is described by Poli et al. (2008, pp. 75–76). In this context, we recommend rewarding partial objectives, like in the case of the forklift truck the successfully transported pallet. Through that, the desired overall objective can be reached in steps. BTs that achieved sub-objectives are kept in the population.

- Many aspects depend on the duration of a simulation run. This factor determines the number of possible episodes and affects the composition and size of the learnable BTs. It also limits the space of possible tasks. Similarly, Poli et al. (2008, pp. 26–27) describe that the time needed for the evaluation of the fitness of one individual is the main limiting factor.
  Applied to the forklift truck scenario, it is impossible to e.g. learn a BT in a task environment with 10 randomly placed pallets, which needed to be collected by the agent. To learn such a task would increase the duration of one trial from days to weeks. An iterative adjustment of the parameters in this period is impractical. Nevertheless, the result of the second learning step shows that the number of pallets can be scaled upwards.

- The granularity of the actions in the gene pool is crucial. They are restricted from two directions. If they are selected too coarse-grained, the range of learnable tasks gets smaller. This can be shown in the scenario of the forklift truck. The composition of the different actions can be arranged in such a

way that pallets are loaded. But in more complex scenarios, in which a direct loading from the front of the pallet is not possible, because of e.g. an obstacle, the learning success depends on the smaller structures of the secondary actions. To learn such cases, a smaller granularity of the actions is needed. But if the granularity is chosen too small, more execution nodes are required to fulfil a task. In consequence, more episodes are needed for the learning process, which increases the total time in dependency on the simulation time. With a smaller granularity, it gets also harder to guarantee safety. The reason for that is that the behaviour in special situations, based on the learned composition of the BT is difficult to predict. Thereby, unconsidered effects can occur, which we showed in the Pac-Man example.

To improve this implementation of a Genetic Programming algorithm, we make two suggestions by following the respective literature, which can be taken up for further work. The first one aims at the simulation used. We suggest adapting the simulation to the task, which should be learned, depending on the granularity of the execution nodes in the gene pool. Iovino et al. (2021) suggest using a Finite State Machine instead of a physical simulation to reduce the computational costs for their pick-and-place scenario, performed by a robot. With a simpler simulation, more episodes are possible, through which the number of execution nodes within the gene pool can be increased. The granularity of the execution nodes and the level of complexity of the simulation should be well-balanced. In this sense, we suggest implementing a simpler, non-physical simulation for the forklift truck to learn other new tasks in a reasonable time of several hours or some days.

The second suggestion aims to improve the setting and optimisation of the Genetic Programming parameters. As mentioned in the first bullet point above, this iterative process can be time-consuming. By using the method of “Bayesian Optimisation”, the process of finding suitable parameters can be automated (Brochu et al., 2010; Rasmussen & Williams, 2008; Snoek et al., 2012). A repository is available, which is implemented in the programming language Python by Nogueira (2014). In a possible next step, the software could be integrated into the existing learning framework.

5 A Cognitive Investigation

In the previous chapters, we presented a learnable agent program for the forklift truck “Crayler”. This includes the learning framework, which is based on Genetic Programming, as well as the resulting BTs. During the implementational process, we observed several aspects concerning the method of learning BTs with Genetic Programming, which are stated in Sections 3.3.4 and 4.3.4. The learned BTs cause a desired machine behaviour within its task environment and were optimised to their fitness function.

In this chapter, we answer the second part of the research question that aims to optimise the AI system towards an intelligent forklift truck. To do so, we investigate the AI system from a *cognitive* perspective. To narrow down the investigation, we specify the AI system as the investigated *object*, define the *level* of the investigation, and identify the *kind* of problem as a problem-solving task. This is done in section 5.1. We then provide the base for the analysis of cognitive functions by dealing with the notion of “*knowledge*” in connection with the forklift truck from different perspectives in section 5.2. This becomes helpful because the notion of “knowledge” is crucial for the design of any agent (Russell & Norvig, 2021, p. 4). In section 5.3, we introduce a *cognitive architecture* that serves as a reference model for human cognition. This model provides a set of cognitive functions that are mainly involved in human problem-solving. We assume that problem-solving by the AI system requires the same cognitive functions as problem-solving by a human. In section 5.4, we investigate, whether these cognitive functions are *executed by* the AI system itself, or by the designers of the AI system during the process of implementation. Those cognitive functions that we assign to the designer point to potential aspects for further improvements. In section 5.5, we outline the properties of an *ideal* intelligent agent as a long-term goal in the context of autonomous *utility machinery* and make further suggestions for improvements of the AI system in this direction by following the respective literature and based on the findings of the cognitive investigation.

5.1 The Subject of Our Investigation

In the following, we detail the subject of our investigation, which includes its task environment. Through that, it is made clear, what we *did* and did *not* investigate. In sub-section 5.1.1, we describe what belongs to the system of investigation and the connection to its environment. This is followed by a categorisation of the task in sub-section 5.1.2, through which it is possible to define the task formally and rationalise it. To focus our investigation and not get lost in the complexity of the system, we introduce the three levels of analysis of information-processing systems by Marr in sub-section 5.1.3. By choosing one of the three levels, we are able to approach the system from a *functional* perspective, which allows us to analyse it regarding the *cognitive functions*.

5.1.1 The Learning Agent Forklift Truck

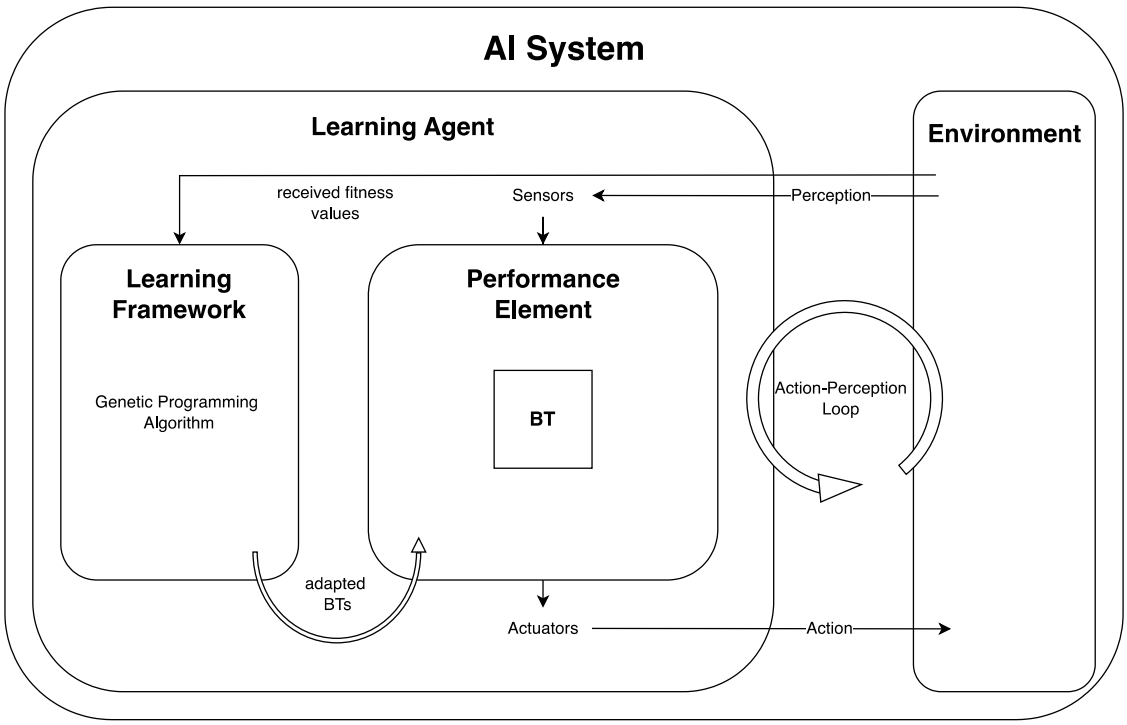


Figure 28: The AI system as a situated system of a learning agent in its environment. Adapted from Russell and Norvig (2021, p. 56, fig. 2.15).

As stated in sub-section 3.1.3, we see the agent in the context of its environment. This means concretely that the subject of investigation is the forklift truck, able to

execute its secondary actions, including the BT control structure (agent program), its defined simulated environment, shown in Figure 22, and the implemented learning framework. We see these components as one connected system and frame it in the following as the *AI system* which is shown in Figure 28.

The visualisation shows the whole AI system and is based on the diagram of a *learning agent* by Russell and Norvig (2021, Figure 2.15). The agent interacts with the environment through its sensors and actuators. The *performance element* is implemented as a concrete BT. The aspect of learning is covered by the genetic programming algorithm, which obtains the necessary values for the fitness function via fulfilled subgoals and thus tests various BTs for adaptation to the task environment.

The learning agent and the environment form an *action-perception loop* through the agent's sensors and actuators. This perspective is not arbitrary and is inspired by the idea of *active externalism* (Clark & Chalmers, 1998). Clark and Chalmers (1998) investigate human cognition, but we assume that their thoughts can be applied in context of artificial agents and their environment. Their core idea is that cognition is not a phenomenon, which only occurs in the brain, but also in the whole body of the agent as well as the environment the agent is situated in (Clark & Chalmers, 1998). Translated to the given scenario that means that the machine behaviour as such does not occur from the CPU only. It needs to be seen in the context of its embodiment, the forklift truck itself, and its environment. E.g. the action "Load Pallet" makes no sense without a pallet in the environment. The agent, in this connection, is domain-specific and coupled to its environment. It follows that we investigate the system as a whole, which connects the agent program and the learning framework together embedded in its environment.

5.1.2 The Kind of Task – A Problem-Solving Task

In Section 4.1.3, we have analysed the task environment with the purpose of selecting an adequate *agent type*. Here, we focus on the characteristics of the task itself to categorise it from a cognitive science perspective. The initial task of the forklift truck is to load several pallets and transport them to a desired location. This type of task is similar, although simpler, to e.g. the task named the "Towers of

Hanoi", because it only puts one item from one to another location, instead of following a specific order of manipulations to reach the goal state. Newell (1979) categorises the "Towers of Hanoi" task as a *problem-solving task*. In this context, he introduces the term of a *problem space*:

> *"Problem Space: A problem space consists of a set of symbolic structures (the states of the space) and a set of operators over the space. Each operator takes a state as input and produces a state as output, although there may be other inputs and outputs as well. The operators may be partial, ie, not defined for all states. Sequences of operators define paths that thread their way through sequences of states." (Newell, 1979, p. 5)*

From that definition, it is possible to formalise a *problem*:

> *"Problem: A problem in a problem space consists of a set of initial states, a set of goal states, and a set of path constraints. The problem is to find a path through the space that starts at any initial state, passes only along paths that satisfy the path constraints, and ends at any goal state." (Newell, 1979, p. 5)*

With this definition, Newell opens up the ability to formalise a problem in the problem space with distinct states. It needs to be mentioned that Newell can be associated with the paradigm of *cognitivism*, which understands cognition as symbol manipulation and information processing. In the following, Newell makes three constraints that define, if an agent (described with "he") can represent a problem in a problem space:

> *"(1) he [the agent] has the space;*
>
> *(2) he [the agent] can obtain representations of the initial states, recognize paths that satisfy the path constraints, and recognize the goal states; and*
>
> *(3) his [the agent's] behavior is controlled so as to attempt the problem in the space." (Newell, 1979, p. 5)*

Applied to the scenario of the forklift truck, we analyse that the problem space is represented by the simulation of the environment and the forklift truck itself. The secondary actions in the space of the simulation are represented by the possible

paths. The goal state[15] is the fulfilment of the task, i.e. the state that all pallets are placed in their slots. Through the BTs, the behaviour of the agent can be controlled in such a way as to attempt the problem. With that, the constraints are fulfilled, so we conclude that the task of the forklift truck *can* be categorised as a problem-solving task.
But Newell (1979) draws attention to another aspect of AI systems. He mentions:

> *"[…] (little man) residing elsewhere in the system, usually off stage, who does all the marvelous things that need to be done to actually generate the total behavior of the subject" (Newell, 1979).*

What he means by that, is that some cognitive functions are taken on *by the designer* of an AI system. We have observed in Section 4.3.4 that the selection of the suitable granularity of the actions and the level of abstraction of the simulation are crucial for a successful implementation of the learning framework. In the genetic programming method we have applied, these selections are taken by the designer and not by the system itself. Applied to Newell's terminology, the designer determines the problem space and with that the problem itself, by choosing the level of abstraction of the simulation. In consequence, the agent embedded in its learning framework cannot generate problems on its own, nor it can solve them entirely by itself. This is shown by the fact that the designer needs to formulate the problem in a fitness function as a mathematical expression.

5.1.3 Marr's Levels of Analysis of Information-Processing Systems

In the two previous sections, we have clarified, what is the subject of investigation and what type of task the AI system should solve. Lieto (2021, pp. 24–26) suggests David Marr's Levels of Analysis of Computational Systems as a suitable tool for the analysis of artificial systems. The concept aims to distinguish between three

[15] In sub-section 4.1.3, we categorised our agent as a model-based, utility-based agent. This categorisation holds as long as the focus is also on the deeper structures of the actions, e.g. the automated actions. For the purpose of our investigation, we are more interested in the functional level of the AI system (see sub-section 5.1.3). In this context, the final state or terminal state, as used by Russell and Norvig (2021, p. 149), is the state at which all pallets are in their slots. For this state, it is sufficient for our purposes to use a binary categorisation, which is satisfied by the term goal state. In this sense, we will use the term goal state in the following.

"[…] levels of organization through which to observe the behaviour of such systems" (Lieto, 2021, p. 24). He mentions that Marr's concept does not provide a tool to analyse general cognition but rather a local cognitive phenomenon, which is in this case problem-solving. He adds that Marr's theory aims to analyse computational systems, which also fits in our case (Lieto, 2021, p. 24). David Marr (2010)[16] suggests analysing an information-processing system on different levels, of which he details three:

1. The level of computational theory
   From the perspective of the first level of Marr's concept, the *functionality* of the different parts of the systems is described. In this sense, only the *cognitive functions* and the *overall goal* are of interest, without investigating the mechanisms beyond (Lieto, 2021, p. 24). The cognitive functions are seen as a mapping from input to output. Lieto (2021, p. 24) describes this level as the "most abstract".

2. The level of algorithms and representations
   The second level focuses on the "how" a task is conducted (Lieto, 2021, p. 24). Here, two aspects are observed. The first one is the *algorithmic* structure, which tackles the realisation of the computational theory on a non-physical, abstract level. The other aspect aims to describe the *representational* structure of the input and outputs.

3. The implementation level
   The last level is concerned with the concrete physical implementation of the algorithmic and representational structure is concerned (Lieto, 2021, p. 24). It tackles questions concerning the *physical* structure and architecture of a system. For the same function on the first level, several concrete physical implementations are possible, which are loosely coupled through each other. E.g. to add two numbers, it is possible to use a simple calculator or a calculator app on the smartphone.

[16] Marr published his idea in 1982. We used the republished version of the book, which was published in 2010 by MIT.

We investigate the AI system of the forklift truck mainly on the first level, which focuses on the general functionality and conception of a system. This goes along with approaching the system through cognitive functions. Partly, we tackle the second level concerning the representation of different aspects, but we do not deal with topics that are located at the concrete implementation on the third level.

5.2 The Notion of Knowledge in the Context of the Forklift Truck

The notion of “knowledge” is a central aspect of designing agents that should handle specific tasks, e.g. transporting pallets from A to B safely. In this sense, two central concepts that are indispensable for designing intelligent agents involve some kind of knowledge. The first one is “rationality”, which uses the agent’s *built-in knowledge* for making rational decisions (Russell & Norvig, 2021, p. 40). The second one is “learning”. This concept is the agent’s ability to gather new knowledge (Russell & Norvig, 2021, p. 41).

Furthermore, the term “knowledge” seems to be inflationary used in the literature, without clarifying the underlying concept explicitly. E.g. Wang et al. (2006) use the term by defining several higher *cognitive functions* like comprehension, learning and problem-solving. In a more applicated technical context of automated BT generating, the term occurs e.g. in connection with Learning from Demonstration (see Section 2.3.3), where it is assumed that a system “acquires” human *expert knowledge* (Iovino et al., 2022, p. 10). Iovino et al. (2021) claim that knowledge about the domain is needed for the learning of BTs with Genetic Programming, but they do not get explicit. Newell uses the term in the context of problem-solving: “All knowledge about a task is obtained by taking steps in a problem space; that is what it means to be working in a problem space” (Newell, 1979, p. 8). In a later work, named “The Knowledge Level”, which we introduce in sub-section 5.2.4, he focuses on this notion entirely, through which the importance of the term is emphasised (Newell, 1982).

We derive from the inflationary use in the literature and the importance of the term for the design of an intelligent agent in connection with problem-solving that it is essential to define the notion of “knowledge” in the context of the forklift truck. M.

Peschl (2021) points to the *functional* perspective in his lecture by making the connection between the term “knowledge” and behaviour clear: “The goal of knowledge is to generate functionally fitting behavior“. In this sense, it is to investigate the agent’s “knowledge” to produce “fitting” or “intelligent” behaviour, as mentioned in the research question.

To approach the term “knowledge”, we start with a formal differentiation from two different perspectives. This is followed by a definition from philosophy. In a more applied context, “knowledge” is considered as a representation in the context of AI. Building on that, we outline Newell’s idea of a “Knowledge Level”. The last approach considers knowledge from the perspective of 4E cognition and includes the vehicle and its environment in the concept. In each of the sub-sections, we describe at first the concept theoretically, to then apply it to the AI system. It needs to be mentioned that this can only be an approach to the notion of “knowledge” from the perspectives of philosophy, AI and cognitive science, which is applied to the concrete example of the forklift truck.

5.2.1 Approaching the Term through Differentiations

As mentioned, we approach the notion by addressing the question of what is knowledge through distinctions. The first differentiation can be best explained by a linguistic expression. The philosopher Gilbert Ryle (2009)[17] suggests distinguishing knowledge at least into two categories. The first one is called “knowing how” (Harre, 2006, pp. 228-229). As the term already includes, this category includes knowledge about how things can be done, e.g. how to make a tea (Harre, 2006, pp. 228-229). This kind of knowledge has an episodic character and is also called *procedural knowledge* (Harre, 2006, pp. 228-229). The second category is called “knowing that” and points to knowledge about facts (Harre, 2006, pp. 228-229). It is also coined *propositional* or *declarative knowledge* and has the property to make propositions about the world (Harre, 2006, pp. 228-229). E.g. knowing that Vienna is a city in Austria. It should be mentioned that for example, Roland (1958) states that this dichotomy is controversially discussed. He also

[17] First published in 1949

mentions that there is a possibility to reduce “knowing that” to “knowing how” under additional conditions, which shows the weaknesses of this distinction.

In the case of the forklift truck, it is possible to apply many aspects to the above definition. The main aspect is the agent’s knowledge about how to load and unload a pallet at the desired slot. This kind of knowledge can be classified as procedural knowledge. The structure in Figure 24 illustrates the representation of procedural knowledge in the form of a BT. It is due to the definition of a BT, as explained in Section 2.1.3 that the actions are executed sequentially. Conditions in the BT, e.g. “All Loaded” in Figure 26 can be categorised as declarative knowledge because they mostly refer to a fact.

Another distinction focuses on the acquisition of knowledge. In that sense, knowledge, especially propositional knowledge, is *a priori* or *a posteriori* (Lowe, 2005). A priori knowledge is independent from the evidence of experience and can be derived (Lowe, 2005). A good example is provided by mathematics. If someone knows that one week has 7 days and concludes that 4 weeks have 28 days. The knowledge of 4 weeks equals 28 days is a priori. This type of knowledge has a deductive character. In contrast to that, posterior knowledge depends on the evidence of experience (Lowe, 2005). E.g. someone knows that the train is arrived at a train station 5 minutes late. This is something which cannot derived a priori and has an inductive character.

Applied to the forklift truck, the agent has a priori knowledge about actions, which are leading to the desired next state for certain. This is only the case if any disruptive factors can excluded. It holds e.g. for the action “Approach Pallet” in the simulation. The agent can be certain that no disruptive factors inhibit its action. Applied to the real world, this does not count anymore and is also visible in the analysis of the task environment in Table 6. Here the task environment is nondeterministic, in contrast to the simulation. The process of the genetic algorithm can be categorised as acquiring a posteriori knowledge since the agent “learns” through trying out. In a wider sense, one could say that the agent makes its “experiences”.

5.2.2 The "Traditional" Perspective in Philosophy: True, Justified Beliefs

The search for a definition of "knowledge" has a long history in philosophy and goes back to Plato's *Theaetetus* (Brendel & Gähde, 2016). Since then, the discussion of the definition has continued over the centuries and focuses mainly on propositional knowledge (Ichikawa & Steup, 2018). Nowadays, a definition of knowledge is discussed in the literature which Brendel and Gähde (2016, p. 11) call the *traditional definition of knowledge*. According to this definition (Ichikawa & Steup, 2018), a subject (S) knows something (p) if and only if:

> "i. S believes that p.
> ii. p is true.
> iii. S is justified in believing that p." (Ichikawa & Steup, 2018, chap. 1)

In this sense, knowledge is defined by the tripartite as *justified, true beliefs*. That means that a subject S needs to believe that something p is true. Additionally, it is necessary that the subject S is justified in believing the p. This gets clear by the following example: Mary is flipping a coin and says, without proof, she knows that it landed tails (Ichikawa & Steup, 2018). By checking, it is true that the coin landed tails, but intuitively we would say that she did not know it. Much more it was a lucky guess rather than knowledge. Because of that, the condition of justification is added to the definition, so that a person needs to justify the belief.

This "traditional" definition of knowledge is criticised by Gettier (1963) and is known as the "Gettier Problem" in the literature. In essence, Gettier shows that true beliefs can be inferred from justified false beliefs (Ichikawa & Steup, 2018). We take the previous coin-flipping example with the assumption that Mary just saw coins, which are similar on both sides. From her perspective, it is a justified belief that the coin landed tails, but it is false. She inferred a justified belief from a false justified belief. The discussion about a solution to this problem in the definition is still going on (Ichikawa & Steup, 2018).

Due to this definition, the forklift truck would have knowledge if it fulfils the upper conditions. In a wide sense, one could argue that the first condition can be fulfilled. We can formulate e.g. that the agent believes that an action leads to a desired goal based on its implemented BT. It is rather questionable if it can justify its belief based

on e.g. modal logic. The agent does not have something like an “introspect” on its knowledge to justify it by itself as mentioned by Russell and Norvig (2021, p. 328). According to the traditional definition of knowledge, we argue that the condition of justification must be inserted by the designer of the AI system in the form of concrete actions and the fitness function to equip the agent with the necessary knowledge.

5.2.3 Knowledge Representation in the Context of AI

Before we access the perspective of “knowledge representation”, we introduce the term “representation”, coming from the viewpoint of cognitive science and AI. Davis et al. (1993) claim that an intelligent entity, including an intelligent agent, can only reason internally. According to them, most things an agent wants to reason about are external, e.g. choosing a rational action for the current environment. Because of this, things in the external world need to be *represented* internally within the agent. In addition to that, Gallagher (2017, p. 84) summarises this so-called “classical concept of representation”, based on Rowlands’s (2006, pp. 5–10) analysis:

> *“1. Representation is internal (an image, symbol, or neural configuration).*
>
> *2. Representation has duration (it’s a discrete identifiable thing).*
>
> *3. Representation bears content that references something external to itself (it refers to or is about something other than itself).*
>
> *4. Representation requires interpretation—its meaning derives from a certain processing that takes place in the subject—like a word or an image its meaning gets fixed in context.*
>
> *5. Representation is passive (it is produced or called forth by some particular situation).*
>
> *6. Representation is decouplable from its current context.“ (Gallagher, 2017, p. 84)*

As described above, the representation of something is e.g. a symbol.[18] A symbol in this sense can be a word or any other token. Russell and Norvig (2021, p. 17) note that these symbols need to be readable by a computer in the context of AI. According to Davis et al. (1993), a collection of symbols, e.g. in a data structure is not sufficient to form what we call “knowledge representation”. There is one ingredient missing. This is a semantic, which corresponds to the real world (Davis et al., 1993). Davis et al. (1993) make this point clear with the following example: A graph, which represents a family structure cannot have cycles, as defined in Section 2.1.1. It is this constraint, which is inherent in the semantics and differentiates between a simple data structure, e.g. a graph and a “knowledge representation” implemented as a constraint graph.

Graphs, which fulfil the upper constraints are also called *semantic networks* (Davis et al., 1993). Because these structures are readable by a computer, they can also be processed. As a special type of semantic network, *knowledge graphs* have developed within the last decades and have become a powerful technical tool (Hogan et al., 2021). Nowadays, knowledge graphs are used by serval tech companies like Google, Facebook, IBM, Microsoft and others (Hogan et al., 2021). This is just one example of applying knowledge as a representation, besides others, e.g. expert systems (Russell & Norvig, 2021, p. 22). The point is, to access the notion of “knowledge” from the perspective of representation turns out to be practical and useful.

In cognitive science, the concept of representation in the form of symbols is connected to the paradigm of *computationalism* (Salisbury & Schneider, 2018). The underlying assumption is that natural intelligence could be explained by symbol processing, similar to a computer (Salisbury & Schneider, 2018). In this context, Newell and Simon formulated their *Physical Symbol System Hypothesis*: “A physical symbol system has the necessary and sufficient means for general intelligent action.”(Newell & Simon, 1976, p. 116). It is the approach to produce intelligent behaviour through the processing of symbol systems.

From this technical perspective of knowledge, the AI system of the forklift truck can be interpreted as a system of symbols on different levels of Marr’s analysis. The

[18] There are also *distributed representations*, which are mostly associated with artificial neural networks and are classified as *sub-symbolic* (Kotseruba & Tsotsos, 2020, p. 26).

controlling BT is a representation in the form of a specific graph. Its action nodes are symbolic representations of the agent's actions, which contain the underlying algorithms. The action node as a diagram, shown in Figure 26 is a representation, which is readable and interpretable to a human and fulfils the six properties of a representation named above. The BT in Figure 26 is a data structure with a semantic that corresponds to the real world because the structure of the sequence of the execution nodes matches a reasonable behaviour of the forklift truck in its environment. With that, it can be concluded that the learned BT of the forklift truck is a knowledge representation. Note that this is not the case for any arbitrary composition of execution nodes. E.g. there is no sense in it, to execute the action "Unload Pallet" before executing the action "Load Pallet".

5.2.4 Newell's Suggestion of the Knowledge Level

As we mentioned in the last section, Newell and Simon (1976) argue that physical symbol systems are necessary and sufficient for intelligent actions and with that for intelligent behaviour. However, they do not mention the notion of "knowledge" in this regard. In a later work, Newell (1982) tries to close this gap. He summarises the understanding of knowledge in a technical way that assumes the structure of representations as the guiding principle, as follows:

> *"If a system has (and can use) a data structure which can be said to represent something (an object, a procedure .... whatever), then the system itself can also be said to have knowledge, namely the knowledge embodied in that representation about that thing." (Newell, 1982, p. 90)*

Roughly spoken that is the statement about what is knowledge in the last section. According to Newell, this perspective has some weaknesses and shifts the issue about AI to the part of the representation. In this context we face our "little man", or alternatively named by Newell "homunculus", of Section 5.1.2 again, when he states: "What is indicative of underlying difficulties is our inclination to treat representation like a homunculus, as the locus of real intelligence" (Newell, 1982, p. 90).

To overcome this issue, Newell (1982) differentiates between the *symbol level*, which builds on several lower levels like (logical) circuits and a distinct *knowledge level*, lying above the symbol level. The symbol level includes the data structure and algorithms of an agent, which is purely mechanical and disconnected from the environment Newell (1982). In contrast to that, the knowledge level is connected to the environment, by following the *principle of rationality*, which includes a directedness on goals. Newell describes knowledge as "[…] a competence-like notion, being a potential for generating action" (Newell, 1982, p. 100). He defines the principle of rationality as follows: "If an agent has knowledge that one of its actions will lead to one of its goals, then the agent will select that action" (Newell, 1982, p. 102). In this sense, the notion of knowledge is defined in a functional way and not structurally like before. In Newell's view, representations are data structures and processes on the symbol level which form the "body of knowledge" (Newell, 1982, p. 100). This also includes that "representation consists of a system for providing access to a body of knowledge, i.e., to the knowledge in a form that can be used to make selections of actions in the service of goals" (Newell, 1982, p. 114). Summarized, he defines knowledge in the following way: "Whatever can be ascribed to an agent, such that its behavior can be computed according to the principle of rationality" (Newell, 1982, p. 105).

Lieto et al. (2018) criticize this approach to knowledge in the context of cognitive architectures and artificial agents by drawing attention to two aspects. The first one concerns the limited scope of such knowledge definition, which is based on representations. According to them, the knowledge is always domain-specific. The second aspect concerns the *homogenous typology*, which can be understood as the problem of processing with different kinds of representations.

In Section 5.2.3, we have argued that the AI system of the forklift truck can be seen as a system of symbol representations. According to Newell, these representations are not knowledge by themselves, rather they form the body of knowledge on the lower symbol level. For knowledge, the principle of rationality is crucial. The learned BT of the forklift truck can be assigned to the symbol level because it matches the properties described above. The BT shown in Figure 26 is purely mechanistic and disconnected from the environment. By adding a goal to the system in the form of a fitness function, the principle of rationality can be applied, so that the agent can rationally compute its behaviour. It is this conglomerate of BT

and goal-directedness, which can be ascribed as knowledge a la Newell. The data structure of the BT seems to assure a homogenous typology within the higher levels of representations. Nevertheless, the critique of Lieto et al. (2018) holds for the limited size of knowledge. The knowledge of the forklift truck is domain-specific and cannot be seen as general at all.

5.2.5 Knowledge in the Context of 4E Cognition

In cognitive science, the so-called “4E Cognition” is a paradigm, which developed over the last decades (Newen et al., 2018, pp. 3–4). The “4E” stands for *embodied*, *embedded*, *extended* and *enacted* cognition. The key concept of this paradigm is that cognition is not an abstract process, which is isolated and is executed in a central processing unit, like the brain (Newen et al., 2018, p. 5). Cognition in this paradigm involves the body, the environment and the situatedness, e.g. the current action. Clark (2012) explains this approach by writing a work with pen and paper. He argues that it is not the case that the whole thinking is done in the brain, but it is also within the writing itself. By reading, e.g. the written words, the externalised thoughts are rethought by the writing person. The written text works in this sense as an external memory, so that the person does not need to remember each of the written words by him- or herself. In this way cognition can be seen as happening in a loop located in the brain, the body and the external environment (Clark, 2012; Clark & Chalmers, 1998).

In the previous sections, the notion of “knowledge” and with that connected “intelligent behaviour” needed some kind of representation. In his article “Intelligence without Representation” Brooks (1991) argues that his mobile robots have some kind of intelligence, without processing representations in a central processing unit. In another article, he makes the connection to “knowledge” more explicit and advises designers of such systems:

“High level abstractions have to be made concrete. The constructed system eventually has to express all its goals and desires as physical action, and must extract all its knowledge from physical sensors. Thus the designer of the system is forced to make everything explicit.“ (Brooks, 1990, p. 5)

Here, Brooks does not define “knowledge” explicitly, but it shows that the notion can have a physical grounding in the paradigm of 4E cognition.

Another example, which illustrates this kind of embodiment more concrete is the so-called “Passive-Dynamical Walker” (Collins et al., 2005). It is the attempt to produce human-like walking, without any kind of computational circuits (Miłkowski, 2018). The “Walker”, which consists of two mechanical legs, coupled by joints, is able to “walk” down a ramp. This is called *morphologic control* and indicates that the control is done by the physical system itself (Miłkowski, 2018). The principle can be pushed further so that the physical structure takes over parts of the computation, which is called *morphologic computing* (Miłkowski, 2018). In connection with the notion of knowledge, the question arises, if the “Walker” knows how to walk. In that sense, “knowledge” can be defined as “embodied knowledge”, which is further discussed by Sorensen and Rebay-Salisbury (2013). From that perspective, “knowledge” is situated in the physical structure of agents. In other words, the designer’s knowledge is constituted in the physical construction of the agent and in the setting of the environment. A radical other approach in this direction is done by Sims et al. (1994), who use Genetic Algorithms to create creatures, made out of cuboids and controlled by simultaneous learned artificial neural networks, in a physical environment. These creatures evolved and developed different behaviours to compete with each other in the physical simulation. The experiment of Sims et al. (1994) shows that it is not only the controlling structure, which causes behaviour but also an agent’s morphology.

In the case of the forklift truck, this perspective tackles an aspect, which is not covered by the other approaches of definitions. It points to the physical body of the forklift truck, which is included in this perspective of knowledge. E.g. the action “Load Pallet” involves a variety of different technical design decisions. This starts with the shape and dimensions of the fork, which needs to match the measurements of a pallet in such a way that heavy loads can be transported safely. It is the whole construction of the vehicle, which is designed for a designated purpose, namely to transport pallets. We argue that the necessary knowledge is embodied within the vehicle’s construction. Similar to the “Walker” one can ask if the forklift truck knows how to load a pallet by its physical construction. We like to mention that it was never the intention of the designer of the forklift truck to insert

morphologic computing within the system. However, the physical vehicle in combination with the learned BT and its underlying structures can be said, to “know” how to transport a pallet. In this sense, the point is that knowledge from the perspective of 4E cognition can be thought of as the physical vehicle in combination with its controlling software embedded in its environment as a whole construct.

5.2.6 The Notion of Knowledge Applied to the Forklift Truck

We argued that the notion of knowledge can be treated from several perspectives. Each of them has something to say about what “knowledge” is and has its own implications. It is for sure that we did not consider all possible perspectives, from which the notion of knowledge can be accessed, but we needed to restrict the selection to a manageable amount.

In Section 5.1.1, we state that we consider the agent of the forklift truck in connection with its physical structure and its environment. We embed the notion of knowledge in the same way. At the beginning of this sub-section, we cited M. Peschl (2021) with the words: “The goal of knowledge is to generate functionally fitting behavior“. By adding Newell’s definition to this statement, the notion of knowledge becomes rationally accessible based on the body of representation. In this sense, we understand knowledge as something, which has a functional property. At the same time, it is inherent within the structure of a BT, in its “semantics”, which controls the agent’s body. According to the perspective of the 4E cognition, knowledge is embodied in the physical construction of the forklift truck in the context of its environment.[19] In this context, we categorise knowledge as “domain-specific” or “domain knowledge”, to define the term, which is also addressed in Iovino et al. (2021).

The initially given categorisations of knowledge make the strengths of BTs explicit to distinguish and deal with procedural and declarative knowledge through different execution nodes. The differentiation between a priori and a posteriori reflects the analysis of the task environment and shows that the learning algorithm gathers

[19] This perspective is similar to the concept of “Affordance”, which is thematized by Andries et al. (2018).

knowledge through a kind of “experience”. Through defining knowledge as *True* Justified Beliefs the hypothesis is supported that the designer, or in Newell’s words “the little man”, with its cognitive functions is crucial for bringing knowledge into the agent and making it work.

5.3 Problem-Solving in Cognitive Architectures

In Section 5.1.2 we applied Newell’s thoughts to the implemented AI system. We argued that there is a “little man” within our AI system, who needs to take control of some aspects of the system to make it work, e.g. defining the problem. The “little man” is a metaphor for aspects of the AI system that are constructed from the outside. We assume that these are the aspects where the agent lacks autonomy. A hint is given by the investigation of the traditional perspective of knowledge according to the forklift truck. We argued that the designer needs to justify the agent’s knowledge. Here it becomes visible that the designer takes the position of the “little man” to make the AI system work. In this sense, the question arises: Which aspects of the AI system are taken over by the designer?
We assume that the question can be tackled by investigating the AI system on the computational level, mentioned in Section 5.1.3. According to the first level of Marr’s categorisation, we need to take a closer look at the involved cognitive functions. Since we analysed in Section 5.1.2 that the kind of task of the forklift truck is a problem-solving task, we are looking for the cognitive functions involved in a problem-solving task.

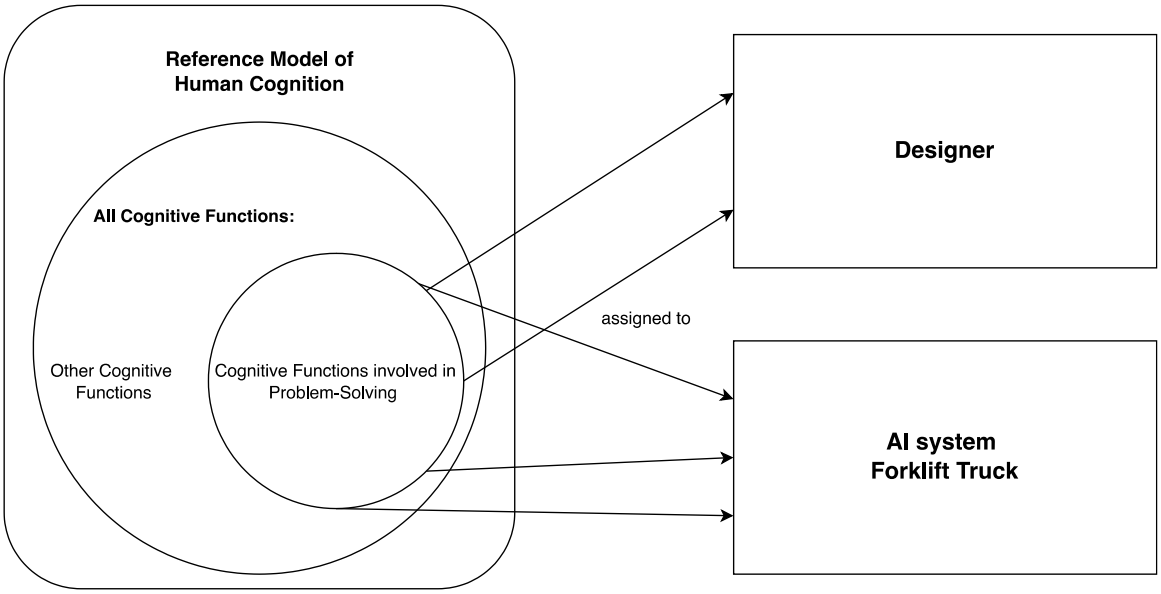


Figure 29: Visualisation of the concept of assigning the cognitive functions, involved in problem-solving, to the AI system or the designer to identify aspects of improvements

In the following, we approach to distinguish between cognitive functions, which are carried out by the AI system of the forklift truck itself and those, which are inherently carried out by the “little man”, namely the designer of the AI system. To do so, we use a cognitive architecture as a reference model, to identify the cognitive functions involved in human problem-solving. This conceptual approach is visible in Figure 29. Cognitive functions that we assign to the designer point to aspects of improvements towards a higher degree of autonomy of the AI system. At the same time, we make the cognitive functions of the AI system explicit.

We give a short introduction to cognitive architectures at first. Here, we present two established cognitive architectures and another cognitive architecture that we use as a reference model for human cognition and which names the cognitive functions explicitly. We shortly explain, how the established cognitive architectures deal with problem-solving to introduce the topic and use their approaches for further suggestions later. The reference model of human cognition makes the cognitive functions, involved in problem-solving explicit. We take those cognitive functions to assign them to the designer or the AI system in the next section.

5.3.1 Cognitive Architectures and the LRMB as a Reference Model

The term *cognitive architecture* goes back to Allen Newell, who argued for “Unified Theories of Cognition” (Lieto, 2021, p. 57). His idea contrasts with the approach of his college Herbert Simon, who tried to tackle small chunks or phenomena of cognition (Lieto, 2021, p. 57). This becomes clear with Newell’s paper in cognitive science: “you can’t play 20 questions with nature and win” (Newell, 1974). In other words, in Newell’s opinion, it is not possible to understand cognition by only investigating distinct phenomena.

Besides that, Newell’s approach is based on the assumption of the analogy between (human or natural) cognition and computer architecture (Asselman et al., 2015). This assumption states that human cognition can be studied by investigating its architecture, similar to the architecture of a computer, concerning its hardware and software components.

Asselman et al. define a cognitive architecture in the following way:

> *„The cognitive architecture is a set of mechanisms of human cognition, which is through the use human knowledge process, specifically it is based on the simulation of human behavior to create a model capable of understanding, reasoning, and problems solving.“ (Asselman et al., 2015, p. 8)*

This definition aims to investigate specifically “human cognition” through simulating behaviour. Besides other phenomena, a cognitive architecture can be used to understand problem-solving, which we try to do in the case of the implemented artificial system.

In their comparison, Asselman et al. (2015) differentiate between three main categories to taxonomise cognitive agent architectures. They suggest, as a first category, *symbolic architectures*, which are based on the paradigms of representationalism and computationalism (Asselman et al., 2015). These paradigms understand cognition as something, which occurs from the manipulation of symbols and their representation. They suggest *emergent architectures* as the second main category (Asselman et al., 2015). The assumption of these architectures is that there is no direct mental representation through symbols. The underlying paradigms are connectionism, dynamical systems, and enactivism (Asselman et al., 2015). In that sense, this approach assumes that cognition

emerges from the adaption through the environment (Asselman et al., 2015). The last categorisation is called *hybrid architectures*, which include elements of symbolic and emergent architectures on different levels (Asselman et al., 2015). Also, Kotseruba and Tsotsos (2020) use a similar categorisation in their survey of cognitive architectures, which emphasises the taxonomy of Asselman et al. (2015). In the context of cognitive architectures, it is worth mentioning two established approaches, which aim to provide unified theories of cognition (Lieto, 2021, p. 63). One of them is named SOAR, which incarnates Newell's idea of cognition (Newell, 1994). The other one is called ACT-R (Anderson et al., 2004). Both of them developed over the years and are categorised nowadays as hybrid architectures (Asselman et al., 2015). Here, we see a trend towards hybrid architectures. We refer the interested reader to the survey of Kotseruba and Tsotsos (2020), in which several other architectures are listed and partly analysed.

To this point, we have seen that cognitive architectures try to study *human cognition*, as stated in the above definition by Asselman et al. (2015, p. 8). Lieto, Bhatt, Oltramari, and Vernon (2018, p. 1) claim that cognitive architectures are useful for AI systems by stating the following:

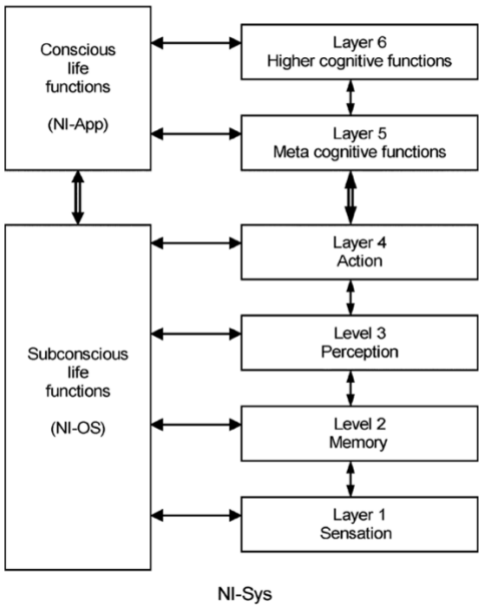
> *"The term "Cognitive Architectures" indicates both abstract models of cognition, in natural and artificial agents, and the software instantiations of such models which are then employed in the field of Artificial Intelligence (AI). The main role of Cognitive Architectures in AI is that one of enabling the realization of artificial systems able to exhibit intelligent behavior in a general setting through a detailed analogy with the constitutive and developmental functioning and mechanisms underlying human cognition." (Lieto, Bhatt, et al., 2018, p. 1)*

By defining cognitive architectures for natural and "artificial" intelligence, the authors bridge the gap for using such models for AI applications like the implemented system of the forklift truck. They claim that intelligent behaviour can be reached by using the analogy to the (cognitive) "functions" of human cognition. In that sense, we are searching for a cognitive architecture, which can be used as a reference for identifying cognitive functions in the concrete problem-solving task of the forklift truck. One possible candidate for such a reference model is the "Standard Model of the Mind", which tries to synthesise three cognitive

architectures, namely: ACT-R, Sigma and SOAR (Laird et al., 2017). The authors analyse that with their model, it is not possible to make statements about metacognition. Furthermore, they do not explicate cognitive functions involved in problem-solving. Because of that, this reference model seems not suitable for our purposes.

Another cognitive architecture, which is not mentioned in the survey of Kotseruba and Tsotsos (2020) seemed more promising. Wang et al. (2006) suggest a descriptive reference model, which is named *Layered Reference Model of the Brain* (LRMB). The authors claim that their model can be used as a reference model for “human” or “natural intelligence”, as they call it (Y. Wang et al., 2006). For their model, they use a bottom-up approach in which they integrate a “comprehensive and coherent set” of cognitive processes[20] (Y. Wang et al., 2006). Based on this model, Wang and Chiew (2010) approach human problem-solving and define cognitive functions, included in the process.

[20] Wang et al. (2006) use the term “cognitive process”. In our case, we are more interested in a functional description of the cognitive processes, so the term “cognitive function” is used. According to Kiely (2014, p. 975), the term “cognitive function” includes mental processes and refers to terms like e.g. decision-making, which is included in problem-solving (Y. Wang & Chiew, 2010). We are aware of the distinction between a process and a function, but both terms point to similar concepts, which are involved in problem-solving. Again the focus is to investigate “artificial" intelligence.



Subconscious Processes		Conscious Processes	
Layer 1	Layers 2-4	Layer 5	Layer 6
Sensational cognitive processes	Subconscious cognitive processes	Meta cognitive processes	Higher cognitive processes
1.1 Vision	2. Memory	5.1 Attention	6.1 Recognition
1.2 Audition	3. Perception	5.2 Concept establishment	6.2 Imagery
1.3 Smell	3.1 Self-consciousness	5.3 Abstraction	6.3 Comprehension
1.4 Tactility	3.2 Motivation	5.4 Search	6.4 Learning
- Heat	3.3 Willingness	5.5 Categorization	6.5 Reasoning
- Pressure	3.4 Goal setting	5.6 Memorization	6.6 Deduction
- Weight	3.5 Emotions	5.7 Knowledge representation	6.7 Induction
- Pain	3.6 Sense of spatiality		6.8 Decision making
- Texture	3.7 Sense of motion		6.9 Problem solving
1.5 Taste			6.10 Explanation
- Salt	4. Actions		6.11 Analysis
- Sweet			6.12 Synthesis
- Bitter			6.13 Creation
- Sour			6.14 Analogy
- Pungency			6.15 Planning
			6.16 Quantification

Figure 30: Schematic representation of the LRMB

On the left side (a): layered structure of the natural intelligence system according to Wang et al. (2006, p. 124, fig. 1). On the right side (b): cognitive processes of the respective layers according to Wang et al. (2006, p. 125, tab. 1).

According to Wang et al. (2006), the LRMB is structured in 6 hierarchical layers as shown in Figure 30. These are building on each other and are interrelated. The first four layers, from the bottom up, are the sensation, memory, perception, and action layers. Wang et al. (2006) denote them in the context of a natural intelligent system as subconscious. The *higher cognitive processes* are denoted as conscious and are the layers including the *meta cognitive functions* and the *higher cognitive functions*. In later works another layer is added in between those two, which contains the *meta inference functions*, so that the model consists of 7 layers in total (Y. Wang, 2010; Y. Wang & Chiew, 2010). A detailed description of the single cognitive processes is given by Wang et al. (2006).

The LRMB as shown in Figure 30 is at first a formal description of human intelligence, as the authors claim (Y. Wang et al., 2006). Partly implementations of the model are based on symbolic representation, mixed with an idea of connectionism, which is visible in their so-called “object-attribute-relation” model (Y. Wang, 2010). According to the taxonomy given above, we categorise the cognitive architecture at the edge of the spectrum of symbolic architectures with a slight connectionistic influence.

According to the statement: “All models are wrong [, but some are useful]” (Box, 1976, p. 792), the question is not, if the model is wrong, but rather, what can the model be used for. We assume that the LRMB can be used as a descriptive reference model for problem-solving. In that sense, we make the assumption that the model is capable of naming the set of cognitive functions involved in problem-solving explicitly in the form of natural language, rather than in a mathematical description. This set is in the following used as a reference on which basis the second part of the research question is investigated. We restrict the model and the derived cognitive functions by excluding the phenomena and concepts of consciousness and emotions.

5.3.2 Problem-Solving in Cognitive Architectures and in the LRMB

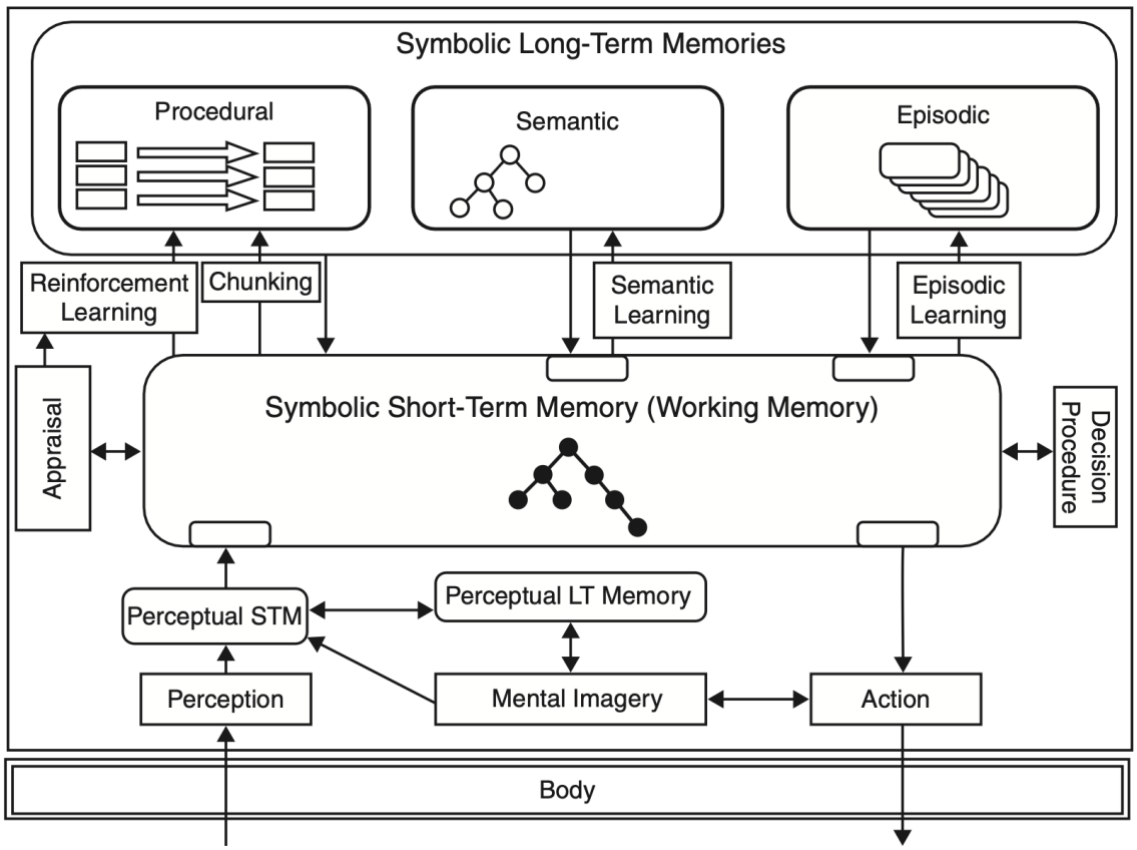


Figure 31: Schematic illustration of the SOAR cognitive architecture

Taken from Lieto (2021, p. 63, fig. 4.1).

In Section 5.1.2, we categorised the pallet-loading task of the forklift truck as a problem-solving task. To formalise a problem, we used Newell's idea of a problem space (Newell, 1979). In the related cognitive architecture SOAR, shown in Figure 31, a problem-solving task is tackled in decision cycles, based on its so-called *Long-Term Memory* (Lieto, 2021, p. 61). This kind of memory is separated into three different categories, *procedural* (if-then production rules), *semantic* and *episodic* (Knowledge about facts) memory, which is inspired by cognitive psychology (Lieto, 2021, p. 61). *Short-Term Memory*, also called *Working Memory* works as a buffer for task-specific knowledge (Lieto, 2021, p. 61). By reading the procedural knowledge in parallel, it is possible to produce cycles of reasoning (Lieto, 2021, p. 61). In a decision cycle, it is checked, if a production rule leads to the desired goal (Lieto, 2021, pp. 61–62). If so, the task-specific knowledge is retrieved from semantic and episodic memory (Lieto, 2021, pp. 61–62). If this is not the case, a subgoal is created, which is then the new goal that must be solved first. This process is known as "universal subgoaling" (Lieto, 2021, p. 62). By adding the found resolution to the existing knowledge, a learning process is realised, which is known as "chunking" (Lieto, 2021, p. 62). It is recognisable that this approach follows Newell's concept of problem-solving through operators in a problem space in a symbolic manner, as described in Section 5.1.2.

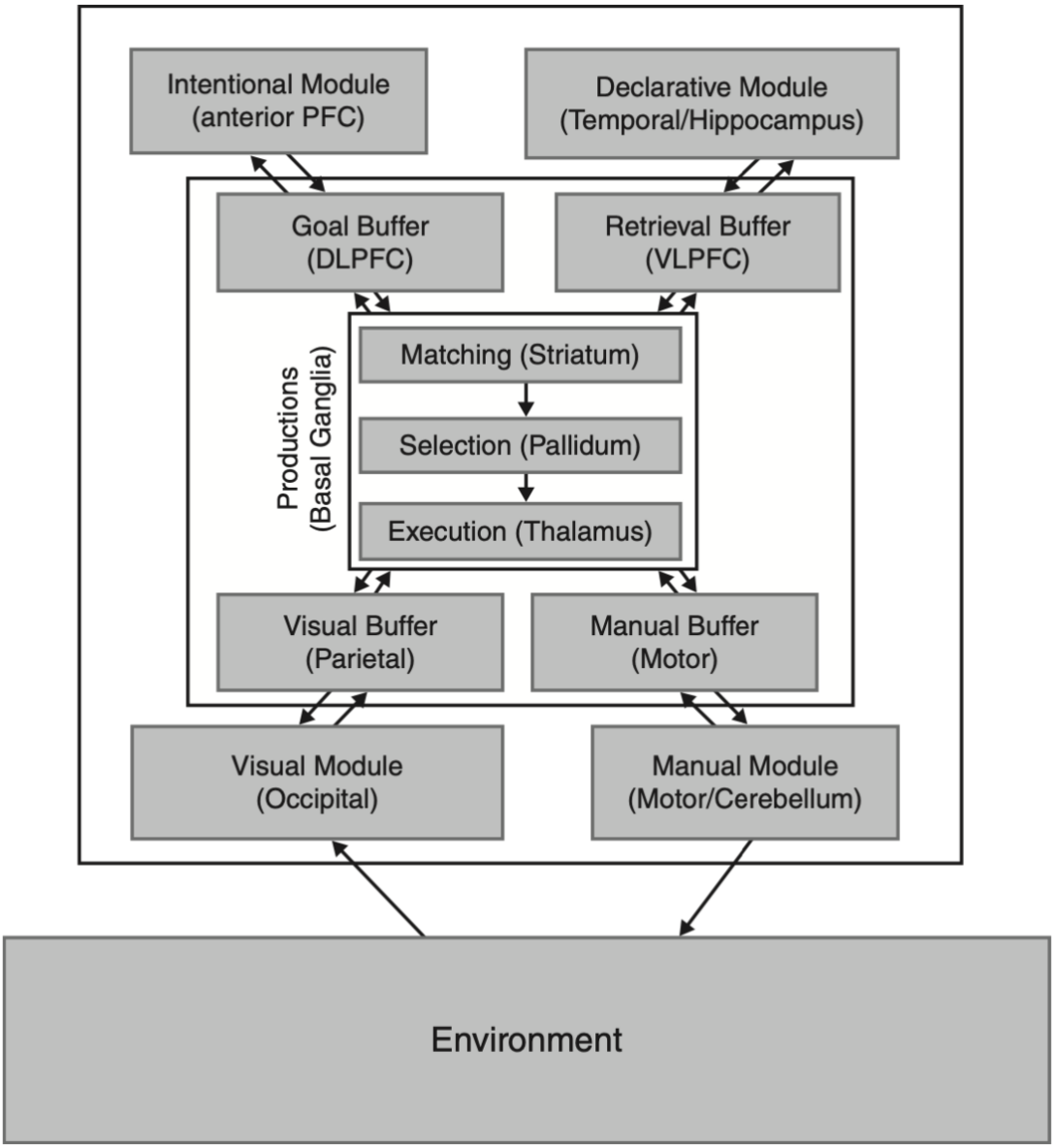


Figure 32: Schematic illustration of the ACT-R cognitive architecture

Taken from Lieto (2021, p. 64, fig. 4.2).

While SOAR is more AI-oriented, ACT-R tries to include insights from psychology and neuroscience as shown in Figure 32 (Lieto, 2021, pp. 63–65). The cognitive architecture consists of four main components, which are connected through a *central production memory* with specialised buffers. The main components are a *visual module* for vision processing, a *manual module* for motion control, a *declarative module* for the memory system and an *intentional module* for keeping track of goals and intentions (Lieto, 2021, pp. 63–65). Within the central production

system, cognitive processing, and in a wider sense problem-solving, is realised by operations of matching, selecting and execution (Lieto, 2021, pp. 63–65).

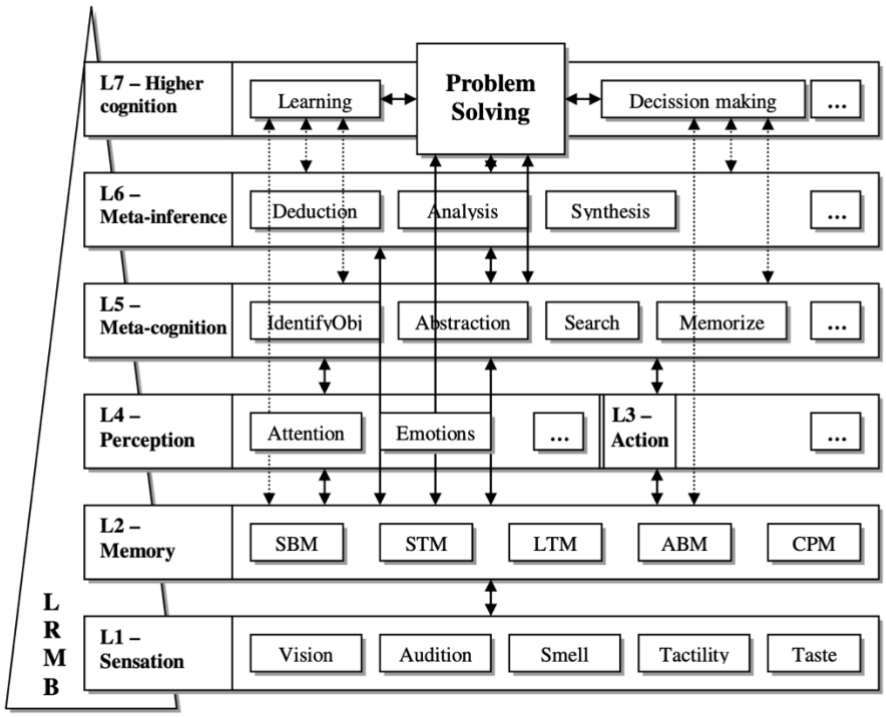


Figure 33: Problem-Solving in the context of the LRMB

The arrows indicate the connection to the other (relevant) cognitive functions, involved in the process. The illustration is taken from Wang and Chiew (2010, p. 90, fig. 5).

Contrasted to these well-established cognitive architectures, we try to approach the problem-solving task of the forklift truck on a descriptive level of cognitive functions. Wang and Chiew (2010) place problem-solving in the LRMB on the level of higher cognition. As visible in Figure 33, problem-solving is not separated from other cognitive functions entirely. Much more, it builds on the functions of the lower levels and is connected to mainly *learning, decision making and comprehension* on the same level (Y. Wang & Chiew, 2010). Wang and Chiew (2010) claim that on the lower levels, the following cognitive functions are explicitly involved, namely *object identification*, *abstraction*, *search*, *memorization* and *inference processes*, such as deduction, analysis and synthesis. Other cognitive processes like

emotions, attention or motivation, besides others, are not considered[21] with respect to an agent that should act rationally (Russell & Norvig, 2021, p. 4).

5.4 Cognitive Functions Involved in Problem-Solving

After we approached the notion of knowledge in the context of the forklift truck, we continued with the investigation of the cognitive functions, which are identified in Section 5.3.2 and involved in problem-solving. The aim is to distinguish between cognitive functions that are performed by the AI system and those that are performed by the designers. The results of this investigation provide the basis for the last section 5.5, which makes suggestions for improvements of the AI system towards an intelligent agent. We would like to recapitulate that this investigation is conducted on the first level of Marr's level of analysis (Section 5.1.3) and is based on the LRMB (Section 5.3.2). The task of the forklift truck is defined as a problem-solving task based on the concept of Newell (Section 5.1.2). We will first define each of the cognitive functions at first and assign them to the AI system or the designer fully or partly.

5.4.1 Identify Objects

Before we investigate the identification of an object, we like to clarify, what an *object* is in the field of AI. Russell and Norvig (2021) mention the term in two different contexts. The first one refers to *physical objects* and goes along with the meaning we have in natural language (Russell & Norvig, 2021, pp. 899–901). The second meaning refers to *mental objects* and points to imaginations, such as concepts or fictional scenarios (Russell & Norvig, 2021, pp. 326–329). We will deal with the first meaning only.

As mentioned in Section 4.1.1, the forklift truck is equipped with visual sensory. Through its implemented software it can perceive pallets and approach them. From the perspective of the forklift truck, pallets are the objects of interest. The visual classifier is able to identify a pallet in such a way that the forklift truck can approach

[21] That does not mean that such approaches cannot be fruitful as well. But we need to restrict this work and follow the suggested selection in the description of Wang and Chiew (2010).

it. In that sense, it can be deduced that the AI system is able to identify the relevant object, namely pallets. Other objects are not identified but are avoided within the path planning. We conclude that the forklift truck has the cognitive function of identifying the relevant physical objects.

5.4.2 Search

The cognitive function “search” occurs in the context of problem-solving. To understand “search” as an embedded cognitive function, we approach it through a suggestion made by Russell and Norvig (2021, pp. 63–64), which identifies four phases for the process of problem-solving, consisting of goal formulation, problem formulation, search and execution.

The first phase aims to set a goal, through which the agent’s behaviour gets organised (Russell & Norvig, 2021, p. 63). Applied to the AI system of the forklift truck, this part of problem-solving in connection with search is implemented by the designer by defining the fitness function in the learning process. This shows that the agent is not able to set goals by itself, nor to adapt them.

Russell and Norvig (2021, p. 64) define in the second phase the formulation of the possible states and actions, which could lead to the goal. In a wider sense that could be understood as the formulation of the problem space, which is presented in Section 5.1.2. By doing so, the problem consists of initial states, and path restrictions, which are caused by e.g. the available actions, and goal states. Through that, the problem becomes rationally accessible. The formulation of the problem space is also placed at the designer’s side. Through the design of the execution nodes within the BT, the designer restricts the possible states and formulates the possible operators inherently. The artificial agent is not able to create new nodes, through which it is not involved in this phase.

The third phase is defined as the actual search within this framework. Russell and Norvig (2021, p. 64) describe “search” within a representation of the environment. Applied to the AI system of the forklift truck, we can distinguish two different kinds of “search”. The former occurs within the learning process. The goal state consists of a possible solution of a functional BT, which is able to load two pallets, like the one shown in Figure 26. On that global level of the Genetic Algorithm, it is about a

functional fitting composition of the available nodes to get to a solution in the duration of one completed simulation cycle. The latter kind of “search” occurs within the BT itself. Through the ticking of the different nodes in the graph, a search is realised, which adapts to the current state of the environment by checking the conditions nodes, e.g. “All Loaded” in Figure 26. In that sense, the phase of “search” can be ascribed to the agent.
In the fourth phase, the found solution is executed. The actions of the forklift truck adapt to the current conditions of the environment, e.g. planning a path from A to B, but they are simply executed as nodes of the BT. This phase is ascribed to the agent as well.
Summarised, the agent is skilled with the pure cognitive function of “search”. Nevertheless, “search” does not occur without a framework, which is in this case the problem space. To realise the cognitive function of “search” as such, a designer is needed, who inserts the boundaries and structure. So we conclude that the AI system is partly capable of the cognitive function.

5.4.3 Inference

The term “inference” is mentioned by Wang et al. (2010) in the context of problem-solving and is represented as a separate level in the LRMB in Figure 33. Because the term is not directly defined by Wang et al. (2010), we approach it from a more general definition, given by the Oxford English Dictionary:

> *“The action or process of inferring; the drawing of a conclusion from known or assumed facts or statements; esp. in Logic, the forming of a conclusion from data or premisses, either by inductive or deductive methods; reasoning from something known or assumed to something else which follows from it […]” (Oxford English Dictionary, 2023).*

By using this definition, it becomes clear that “inference” is connected to reasoning and concludes from given data to something else by using methods of induction or deduction. Wang et al. (2010) mentions that “inference” acts based on represented knowledge.

In the context of problem-solving, the result of “inference” can be thought of as possible paths in the problem space, which lead from the initial state to the goal state. Wang et al. (2010) describe this cognitive function as deductive. Deductive reasoning takes general premises and infers from them a conclusion within a logical framework. In AI this can be achieved by e.g. condition-action rules, like in the form: “if these conditions are ‘true’, then do something”. Through that, an agent can navigate from state to state in the problem space to get to solutions through inference. Wang et al. (2006) define “analysis” as a cognitive function, which divides an object into its parts and investigates its relations in a deductive manner. In contrast to that, they define “synthesis” as a cognitive function, which gathers objects and concepts to a larger ensemble in an inductive way. Induction tries to derive general statements of rules from specific premises, which are based on e.g. experience. We mentioned this in the context of knowledge in Section 5.2.1.

Applied to the forklift truck, we can identify deductive reasoning on the side of the AI system. As we mentioned in Section 2.2.1, a BT includes classical condition-actions rules. This deduction is based on the agent’s knowledge, which is partly represented by the BT. Within this explicit symbolic structure, deductive inference is possible, e.g. by checking the condition “All Loaded” in Figure 26 before the action sequence is repeated. As mentioned in Section 5.2.1 the process of learning through the Genetic Algorithm can be seen as inductive reasoning, because knowledge is gathered a posteriori. In difference to that, the agent is not able to “analyse” nor “synthesis”. It e.g. does not have the competence to divide its action nodes into smaller pieces, to adapt them to a changed environment. E.g. more complex situations, like a pallet being blocked by another need a finer granularity of the possible actions to learn such a task. In conclusion, the agent has the principle ability of deductive and inductive reasoning within the limitations of its knowledge representation, namely the BT. However the higher cognitive functions, such as “analysis” and “synthesis” are performed by the designer of the AI, so that the AI system is partly capable of the cognitive function.

5.4.4 Memorization

Wang et al. (2010) mention “memorization” within their visualisation of cognitive functions, which are involved in problem-solving as shown in Figure 33. In this context, it is the process of storing solutions to e.g. reuse them. Applied to the problem space, a solution, which should be memorized is a possible path from the initial state to a goal state. For the process of problem-solving, it is necessary to “remember” the steps in between. If the agent finds an action, which leads to a state closer to the goal, it needs to “remember” the action and the next state to finally find a suitable solution. The former is needed in the long term, while the latter is used in the short term. Wang et al. (2006) define “memorization” at the level of meta-cognition with the properties of encoding, storing and retrieving information. With this cognitive function, it is not meant the locus of memory rather than the process. According to Wang et al. (2006), the different kinds of memories are located on the lower second level of the LRMB.

Applied to the forklift truck, we can deduce that the AI system is able to “memorize” specific things. This is shown by the learning process. In Figure 24 a sequence of actions to load one pallet is visible. Due to the properties of a BT, the temporal order of the sequence is fixed. As we analysed in Section 5.2.1 this can be categorised as procedural knowledge or knowledge of “how to do something”. By storing the sequence of actions within the framework of a BT, the agent is able to “memorize” procedural knowledge. The memorization of facts or states is needed in the short term. This is e.g. the case of using a blackboard within the AI system as mentioned in Section 4.2.4. It can be concluded that the AI system is able to “memorize” within the framework of BTs.

5.4.5 Abstraction

Wang et al. (2010) localise the cognitive function of “abstraction” at the level of meta-cognition. Within the LRMB, Wang et al. (2006) describe that “abstraction” is involved in generating models and concepts of the world as representations. Russell and Norvig (2021, p. 66) define “abstraction” as “the process of removing detail from a representation […]”. In the context of problem-solving, they ascribe to this process the functional property of describing a problem suitable. We can

embed this conceptualisation into the framework of a problem space. In this sense, it is the formulation of the initial and goal states, as well as the states in between and the operators with an accurate level of detail. This becomes more clear with an example. If an agent likes to travel from Vienna to Basel, the cities and those in between can be represented by nodes in graph. The connections between the nodes are the possible train connections. In this model, the level of detail is suitably chosen to find a path leading from Vienna to Basel via train. It is not part of the problem, how the train stations look like or if it is possible to buy a coffee there. Russell and Norvig (2021, p. 66) note that an intelligent agent would be swamped by the richness of detail in the real world without the ability to abstract.

The notion of “abstraction” seems to be similar to the previously mentioned term “granularity” in Sections 2.1.5, 3.3.4 and 4.3.4. From the article, written by Wang et al. (2017) follows that problems in the real world can be accessed from different levels of granularity. In contrast to “abstraction”, the concept of “granularity” can be seen as the level of distinguished parts of a whole. In that sense, a coarse granularity includes all its subparts without removing its details, compared to the concept of “abstraction”. While “abstraction” is a cognitive function with a focus on the degree of detail of information, the term “granularity” describes the level of the fineness of observation.

Applied to the forklift, we can distinguish two different areas, in which the “abstraction” and “granularity” are crucial. The first one is the level of the granularity of the possible actions, which is mentioned in Section 4.3.4. Here it is the designer, who needs to make an appropriate choice according to the problem. The second one is the choice of the level of abstraction of the used simulation environment for the learning algorithm. By e.g. choosing a physical simulation of the environment and the forklift truck, there is a high degree of detail included. The aim at this point is, to match the level of abstraction of the simulation with the problem to solve. Thought in the concept of the problem space, this means to design it as small as possible and as large as necessary. However, it is a choice, which is made by the designer and not the forklift truck, so we can conclude that the AI system is not able to “abstract”.

5.4.6 Connection to Higher Cognition

According to Wang et al. (2010), the cognitive function of problem-solving is connected to others on the level of *higher cognition* in the LRMB. They explicitly name “comprehension”, “decision making”, and “learning”.

The authors describe the cognitive function of “comprehension” from the perspective of cognitive psychology as the capability to connect a new representation of something with existing knowledge (Y. Wang et al., 2006). This includes not just an extension of the existing knowledge representation, but also further connections to this base. In other words “comprehension” can be understood as the embedding of new representations into the existing knowledge base of an agent. In this way, a new problem can be solved based on partly existing knowledge. In the case of the forklift truck, this does not happen. A problem task is solved through the Genetic Algorithm. Within this process, a new BT evolves, but it is not extended.

Wang et al. (2006) describe the cognitive function of “decision making” as a selection of a set of actions. Russell and Norvig (2021, pp. 528–529) define “simple decision making” in a similar way. They add the notion of a *utility function*, which includes the agent’s “preferences”. Wang et al. (2006) describe that these "preferences” of rational agents lead to rational decisions. To find a solution for a given problem the selection of actions is indispensable. In the case of the forklift truck, “decision-making” is done in two different steps. The former is the selection of the action sequence through the Genetic Algorithm. The agent’s “preferences” are captured by the fitness function. The latter is the selection of the current action, which is executed in the BT. Here, the agent’s “preferences” are inherent within the structure of the BT.

The notion of “learning” is defined by Wang et al. (2006) as gaining knowledge or the acquisition of skills. In the context of a rational agent, Russell and Norvig (2021, p. 41) built on the distinction, between a prior and a posterior knowledge, which we presented in Section 5.2.1. In the case that the environment of the agent is completely known, the agent has the a prior knowledge it needs, through which “learning” is not necessary. If the environment is unknown, the agent learns through its experience in connection with the environment. To solve a problem, “learning” is needed to gain this additional a posterior knowledge. In the case of the forklift

truck, the AI system gains this knowledge through the learning algorithm, as we have shown in Section 5.2.1.
We can conclude that the AI system is not able of the cognitive function “comprehension”. The "decision-making" is partly taken over by the agent, but the "preferences" must be defined by the designer. The AI system is able to learn, in the sense of a rational agent, which means it can gather posterior knowledge.

5.5 Towards an Intelligent Agent

Based on the analysis in the previous section, we like to connect the investigated cognitive function with the notion of an *intelligent agent*, applied to the example of the forklift truck. To do so, we clarify the term “intelligence” in the light of functional fitting behaviour, according to a respective environment and its goals. This is followed by a clarification of which cognitive functions are performed by the AI system and at which points the human designer interacts indispensably. The last section points out the weaknesses of the AI system and makes guiding suggestions to improve the implemented system.

5.5.1 Guiding Principles for Ideal Intelligent Agent Design

To classify the notion of “intelligence” in the wide range of different definitions from a variety of disciplines, we like to approach the term more general at first, then get more precious with a focus on AI. Legg and Hutter (2007) collected 70 definitions from different disciplines like psychology and AI research to synthesise the decisive properties of these definitions. They conclude that “intelligence”:

- *“Is a property that an individual agent has as it interacts with its environment or environments.*
- *Is related to the agent’s ability to succeed or profit with respect to some goal or objective.*
- *Depends on how able the agent is to adapt to different objectives and environments.” (Legg & Hutter, 2007, p. 9)*

Their conclusion shows that “intelligence” is directed to a goal situated within an environment. The ability to adapt to different environments and goals is a decisive characteristic of "intelligence".

In a technical setting, the question arises, of how to create “artificial intelligence”. Russell and Norvig (2021, pp. 1–4) propose two dimensions to approach AI. These dimensions cover the underlying assumptions about what “intelligence” is. The former distinguishes between “intelligence” as a property of *human performance* and a more abstract and formal way of defining it through *rationality* (Russell & Norvig, 2021, p. 1). A possible definition of rationality is given by the principle of rationality in Section 5.2.4. The latter distinguishes between focusing on “intelligence” as a *process of thoughts and reasoning (thinking)*, and *(intelligent) behaviour*, which is associated with *acting* (Russell & Norvig, 2021, p. 1). With these two dimensions, four different approaches to AI can be derived: *thinking humanly*, *acting humanly*, *thinking rationally*, and *acting rationally*. Russell and Norvig (2021, pp. 1–4) note that the different approaches help each other by exchanging knowledge from empirical approaches e.g. psychology and rationalist approaches, which use disciplines like mathematics and engineering.

By following the research question, the interest is in a technical application. In that sense, it is a matter of the “right” behaviour in the quantity of environments. By following the previous work at the AIT, we like to state our assumption clearly that we followed the path of intelligence as rationality. To conclude, “artificial intelligence” in this work is seen as acting rationally.

A BT as used in the AI system follows exactly this approach. It controls the behaviour of an agent in a rationalised way through symbol representation and manipulation within the structure of a tree. Our approach of using the LRMB to identify the different cognitive functions, which are involved in problem-solving, can be interpreted as support from the direction of acting humanly because the cognitive architecture aims to model human cognition and with that human problem-solving.

Russel and Norvig describe *acting rationally* with the simplified words: “*AI has focused on the study and construction of agents that do the right thing*” (Russell & Norvig, 2021, p. 4). In this context, “artificial” intelligence is connected with rationality as stated before. The entity, which behaves in an environment is the agent, as explained in the introduction. A rational agent “[…] acts so as to achieve

the best outcome or, when there is uncertainty, the best expected outcome" (Russell & Norvig, 2021, p. 4). To act rationally, it is necessary to provide the agent with some sort of objective or goal, which is called the *standard model* (Russell & Norvig, 2021, p. 4). Within the triangle of a rational agent, its environment, it needs to adapt to, and an objective or goal, an ideal *intelligent agent* can be defined as taking "the best possible action in a situation" (Russell & Norvig, 2021, p. 34). The "best possible action", in that sense, is oriented on the respective goal.

There are different approaches to implementing an intelligent agent. To determine the behaviour of an agent by a BT can be classified as "*Good Old-Fashioned AI*", which is categorised by capturing the "right" behaviour in logical rules, e.g. condition-action rules[22], through the processing of representations (Russell & Norvig, 2021, p. 982). The underlying paradigm is computationalism, which we introduced in Section 5.2.3. This approach has already been criticised by Alan Turing (1950). He formulates "the Argument from informality of Behaviour", in which he states that it is not possible to capture the complexity of behaviour in a set of (logical) rules. From this critique, we follow that an approach should cover behaviour more informally. Russell and Norvig (2021, p. 982) mention in that regard *probabilistic reasoning systems* and *systems of deep learning* to deal with rather "informal" tasks.

By thinking about AI, often the question arises, how similar is the "intelligent" machine to a human? To tackle this question, we introduce the so-called "Chinese Room Argument", which was proposed by the philosopher John Searle (1980). In his thought experiment, he imagines himself in a locked room with a translation dictionary in the Chinese language. He does not know any Chinese. Nevertheless, he translates a script that he receives and sends the translations back again. From the outside, it looks like he "knows" Chinese, because he can translate it, but it is

[22] We mentioned in Section 2.2.1 that BTs are a structure that can capture behaviour not just as condition-action rules, but also through its hierarchical order, the variety of nodes, and its execution rules within the tree structure. Nevertheless, the implemented AI system does not involve probabilistic reasoning (Russell & Norvig, 2021, pp. 385–646) nor deep learning techniques (Russell & Norvig, 2021, pp. 750–789), except for object identification, which is implemented as a previously trained artificial neural network. Theoretically, the implemented BT can be reduced to a very complex set of condition-action rules, through which it can be categorised as Good Old-Fashioned AI. We like to note that BTs can be expanded by specified nodes, which can involve e.g. probabilistic reasoning or deep learning techniques. Through that BTs can provide the framework for hybrid control structures, similar to hybrid forms of cognitive architectures, which we mentioned in Section 5.3.1.

clear that he cannot distinguish between a Chinese sign and a Japanese sign. He just used the rules for translation in the dictionary. Searle (1980) concludes that this kind of (artificial) intelligence only pretends human thinking or behaviour and names it *weak AI*. Russell and Norvig (2021, p. 981) describe this weak AI as a machine that pretends to "act as if it is intelligent", while it is not consciously thinking. In contrast to that, Searle (1980) denotes *strong AI* as a human-like intelligence, which includes conscious states and thinking. Russell and Norvig (2021, p. 981) mention that strong AI is also called a "*general AI*" or "*human-level AI*" nowadays. Because of this distinction, we decided in Section 5.3.1 to exclude the phenomenon of consciousness from the used reference model. The AI system of the forklift truck can only be a weak AI, but we assume that it can be inspired by a comparison with human intelligence.
To conclude, we have stated, which kind of AI is forced regarding the forklift truck. This kind of AI and its approach is denoted as an intelligent agent and defined above. Together with the more general definition of "intelligence" at the beginning of this Section, we list the following properties and qualities as guiding principles for a functional AI in the context of the forklift truck:

- The principle of rationality in connection with a goal or objective
- The adaptability to a set of different environments and different goals or objectives
- The implementation of a weak AI, which includes the absence of any kind of consciousness or emotional states

This list needs to be extended by two aspects, which are derived from practical reasons and are mentioned in Sections 2.2.1 and 4.2.4:

- Fulfilling safety guarantees
- Human readability and understandability to prevent unintended behaviour

Together, these properties define, how the ideal intelligent agent should be. Through this definition, it is possible to qualify the direction of possible suggesting

improvements, which are deduced from the next section and connected in Section 5.5.3.

5.5.2 Cognitive Functions of the AI System

In the previous section, we defined the notion of an “intelligent agent” and categorised the used approach to build an AI for the forklift truck. From that, we derived guiding principles for an intelligent agent. Based on the analysis of section 5.4, which is supported by section 5.2, we deduce, which cognitive functions the AI system is capable of. The findings are used in the next section to make suggestions for improving the AI system towards an intelligent agent according to the guiding principles.

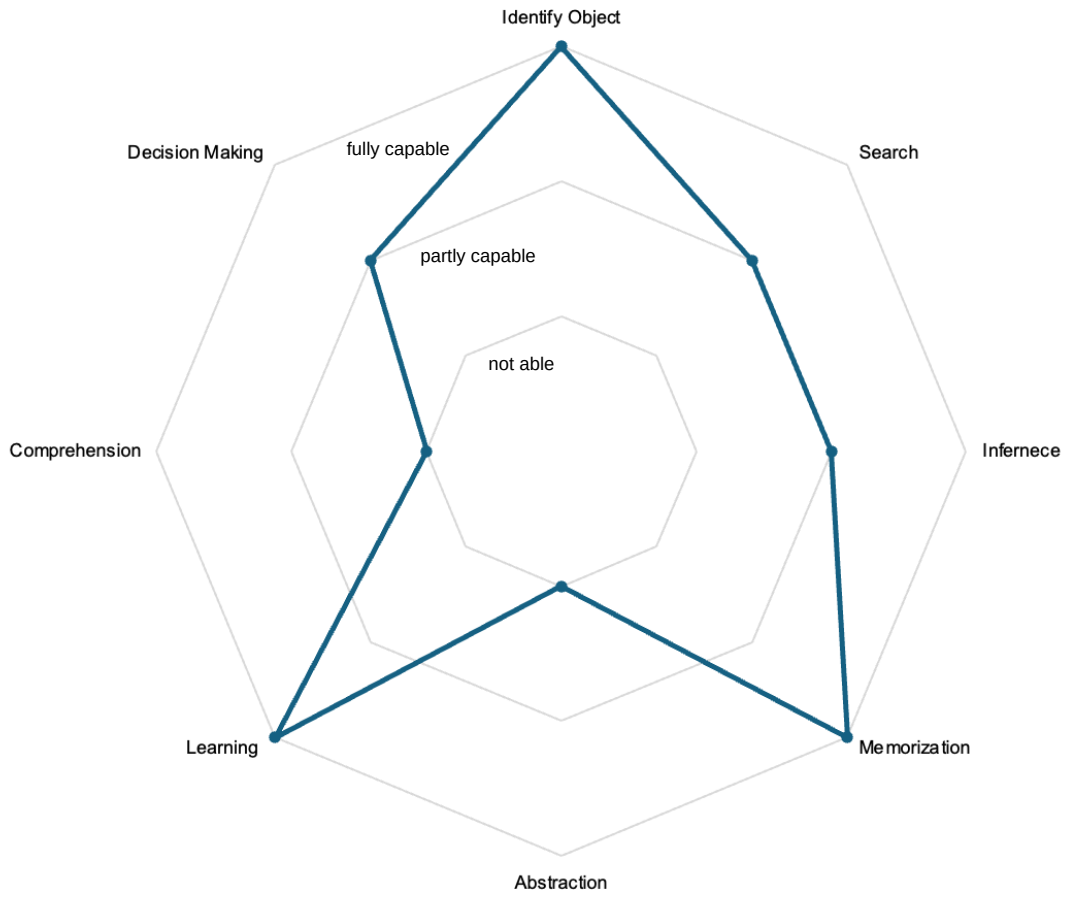


Figure 34: Radar chart of the cognitive functions involved in problem-solving

The chart shows in three categories (not able, partly capable, and fully capable) the abilities of the AI system of the forklift truck, based on the analysis of section 5.4.

To visualise and conclude the findings of the analysis in section 5.4, we assigned the AI system's capabilities regarding each of the eight cognitive functions into three qualitative categories. These are: *not able*, *partly capable*, and *fully capable* of performing a specific cognitive function. The categorisation "not able" means that the AI system is not capable of the respective cognitive function. In these cases, the human designer must perform the cognitive function to make the system work by e.g. setting the system boundaries, constructing the representational space, like a BT, and defining goals. This is the case for the cognitive functions "abstraction" (Section 5.4.5) and "comprehension" (Section 5.4.6), which is visible in Figure 34. If there is any such thing as a "little man" in an AI, which is mentioned by Newell and described in Sections 5.1.2 and 5.2.4, it is to be found at this point.

A cognitive function of the AI system is categorised as "partly capable" if the main function is conducted by the agent itself, but within a constructed and restricted framework. In other words, the cognitive function can only be performed by the AI system with the heavy support of the designer and mostly in a domain-specific setting. This gets more clear with the cognitive function "search" (Section 5.4.2). The pure search operation is conducted by the system itself but is heavily constrained and logically determined by the designer's implementation. We assigned the cognitive functions "inference" (Section 5.4.3) and "decision making" (Section 5.4.6) in this category as well.

In the category of "fully capable" are those cognitive functions, which are conducted by the AI system itself and are assumed to be adaptable to a variety of different environmental settings. The cognitive function of "identify object" (Section 5.4.1) of the forklift truck can be used as an example. The AI system is able to detect pallets as mentioned in Section 4.1.1. This is not restricted to one very specific environment. The forklift truck can perform this cognitive function at the testing area just as in another place. Here, the designer does not need to define any other domain-specific constraints, nor set a specified framework. Other cognitive functions, which are assigned to this category are "learning" (Section 5.4.6) and "memorization" (Section 5.4.4).

We can conclude that several cognitive functions are conducted by the designer. These are those domains in Figure 34, of which the AI system is not "fully capable". At the conceptual level, this is the construction of the representation framework,

the definition of goals and sub-goals and the harmonisation of the degree of abstraction and granularity between the AI system and its environment. Applied to the forklift truck in concrete, this is the implementation of the BT, the formulation of the fitness function as a specification of a goal and its subgoals, and the selection of the granularity of the actions as well as the level of abstraction of the simulated environment. The required processes are made explicit through the analysed cognitive functions and the visualisation in Figure 34.

5.5.3 What is Missing Towards an Intelligent Forklift Truck

In the following section, we conclude our results, connect the lessons learned in the previous chapters with insights from this chapter, and make further suggestions to guide the current AI system towards the direction of an intelligent agent system, which we specified in the previous section. To do so, we use the identified lacking cognitive functions as an analytic tool for suggesting further improvements.
First, we recapitulate the findings from Chapters 3 and 4, which are based on our observations as a designer of the AI system. In Section 3.3.4, we have summarised the lessons learned from the Pac-Man benchmark example. These can be summarised as follows:

- To design the actions and conditions for the agent Pac-Man, the designer needed to implement domain-specific knowledge into the execution nodes. This becomes explicit by designing the action “Approach Big Pill”, which needed to be modified and added with more information.
- The fitness function has special importance. If it is not well-defined, the learning algorithm past the desired task to be learned.
- The implementation of the Pac-Man game needed to be adapted to optimise the simulation time of the learning algorithm. This was done by foregoing the visual display on the screen, which can be interpreted as abstraction. The visual display of the game is important for a human but not for a machine.
- The new combination of execution nodes caused by the learning algorithm can lead to unpredictable behaviour of the agent. This becomes crucial concerning safety constraints.

In Section 4.3.4, we stated the observations of the implementation of the Crayler learning framework and made the first suggestions for improvement, which can be summarised as follows:

- The designer needs a rough idea of how the learned BT should look like, especially concerning its size. This is important to adjust the costs within the fitness function and the control parameters of the Genetic Algorithm.
- The fitness function needs to describe the task as precious as possible.
- The granularity of the execution nodes determines the complexity of the learnable tasks, the size of the required BT and the necessary time to learn, through the combinatorically dependence.
- We suggested adapting the abstraction level of the simulation to fit the granularity of the execution nodes to reduce the simulation time.
- We suggested to automate the process of optimising the parameters of the Genetic Learning algorithm by using Bayesian Optimisation.

In section 5.2, we approached the term “knowledge” within the AI system of the forklift truck. We analysed that the AI system has “knowledge” in the form of symbolic representation in the BT, which provides the base for cognitive functions like e.g. inference, as we argued in Section 5.4.3. Together with a specified goal, set by the designer, the principle of rationality can be applied, so that the AI system can be attributed to have “knowledge” in the sense of a knowledge level a la Newel. Further, we argued that there is embodied knowledge, which is inherent in the design of the forklift truck itself in connection with its environment and situation. From the definition of “knowledge” as true justified beliefs, we have seen that the AI system is not able to justify and a designer is required. We distinguished between declarative and procedural knowledge and showed that a BT can represent both types. New knowledge can be acquired through the learning algorithm in the form of a posterior knowledge.
We have seen that knowledge has a property of directedness towards e.g. a goal. This is also reflected by Newel’s formulation of a knowledge level by connecting his definition with the principle of rationality, which is by itself goal-directed. As we mentioned before, Peschl (2021) gives knowledge the purpose of producing

“functionally fitting behaviour”. In this sense, knowledge can be seen as the base on which the different cognitive functions operate to produce behaviour. We argued for this connection, e.g. by analysing the cognitive function of inference within the forklift truck. Here the process of inference takes place within the BT structure, which we categorised as a knowledge representation. Other cognitive functions like learning and memorization are also highly connected with the BT.

We analysed, as shown in Figure 34 that the AI system is not able to perform the cognitive functions of abstraction and comprehension, which answers the second part of the research question. The absence of abstraction could explain some of the above observations during the process of implementation. We observed that the formulation of the goal, in the form of a fitness function, and the degree of detail of the simulation together with the level of granularity of the execution nodes is crucial for the successful learning of BTs. To get closer to, what we define as an intelligent agent, we suggest setting a focus on the mentioned missing cognitive functions in the AI system.

A form of abstraction could be realised by an automated sub-goaling, similar to the idea of “universal subgoaling” and “chunking” in the cognitive architecture SOAR, described in Section 5.3.2. The idea is that the AI system sets its own subgoals in relation to an overall goal. Within the sequence of loading a pallet, a subgoal could be e.g. to approach a pallet correctly by having it on the fork. The possibility of structures in the form of PPA, which are described in Section 2.1.5, could support this approach. Applied to the problem space, a subgoal would be a new state, which is closer to the goal state than the previous state. Through this approach, the AI system would be able to define these closer states. If more complex situations occur, like a pallet cannot be accessed because it is blocked by another and the default action sequence as shown in Figure 24 cannot be executed, the AI system could formulate and check its subgoals. In the case that one of the subgoals cannot be reached, the AI system could identify this coarse-grained action. In the next step, the AI system could e.g. try to learn this new situation by using a Genetic Programming algorithm within a simulated environment. Therefore, the available execution nodes in the Gene Pool are the finer-grained ones of the identified action. For sure, safety needs to be granted for the learned BT. In this way, something that we call *artificial abstraction* could be realised, where

the level of the required detail is set through a top-down approach by the formulation of subgoals.

By adding these newly learned BTs to the existing default BT, the AI system would be capable of including the cognitive function of comprehension. This kind of comprehension is based on the BT structure. It follows the definition given in Section 5.4.6, which understands comprehension as the embedding of knowledge on a representational level. By adding a subtree to an existing BT, the inherent knowledge is connected through the structural functionality of the BT itself, e.g. the subtree is only executed under specified conditions.

Connected to this approach, it is also conceivable to include execution nodes or whole subtrees with different learning approaches. We have mentioned several variations of BTs in Section 2.1.4, e.g. nodes with active inference or stochastic subtrees. This could be useful for the AI system to adapt to a variety of different task environments. Russell and Norvig (2021, p. 982) support this assumption if they state that probabilistic reasoning systems and deep learning approaches are suitable for other tasks than “Good Old-Fashioned AI” fits for. Similar to the trend of hybrid cognitive architectures, which we described in Section 5.3.1, BTs could include emergent elements within their framework to reproduce a wider range of behaviour. Through their property of being modular, BTs provide a suitable framework for a hybrid architecture. Because it is possible to execute different subtrees of a BT logically separated from each other, emergent or stochastic modules can be implemented under conditions with special safety regulations. Through that, the additional aspects of an intelligent agent can be fulfilled, namely being safe.

In Section 4.1.3, we have analysed the task environment of the forklift truck and detected that the real-world scenario is complex. Based on this we suggested a goal-based, model-based agent design, which is directed towards a specific goal. Our observations during the implementation of the learning algorithm show that more complex scenarios with blocked pallets are difficult to achieve. Through the analysis of missing cognitive functions, we could identify possible weaknesses and key competencies of the AI system and come up with adapted suggestions. In conclusion, we suggest using a hand-made BT as a ground structure for the control of the forklift truck. Based on that the suggestions above can be integrated, as well

as other emergent approaches, to realize a more adaptive behaviour in the sense of the guiding principles for an intelligent agent.

6 Discussion

In the following, we discuss and reflect on the results of this work. To do so, we start with a discussion about the first, more technical and applied part, which includes Chapters 2, 3, and 4. Here the focus is on the technical applicability of the implemented learning framework and the use of BTs in the context of utility machinery on the concrete example of a forklift truck. This is followed by focusing the discussion on the second part, which takes the system of the forklift truck as a subject of investigation from a theoretical perspective in the context of cognitive functions and concepts, which are inspired by a variety of disciplines and fields like cognitive science, philosophy or computer science. At the end, we give an outlook of possible further research.

6.1 Summary and Discussion of the Technical Aspects

The decision to use BTs as a structure to control and represent behaviour was forced by the researchers of the AIT. Pro arguments for using BTs in the context of utility machinery are their modularity, hierarchical structure, reactivity, and human readability, which are stated in Section 2.2.1. Against the usage of BTs are concerns in connection with the complexity of the implementation of BTs, the tendency of overfitting, and the tendency to check a large number of time-consuming conditions, which is stated in Section 2.2.2.

We have decided to use Genetic Programming as an approach for learning BTs. The decision was based on the insights of Iovino et al. (2021) and Zhang et al. (2018), who applied the method to both, the setting of a Pac-Man environment and an industrial robot application. Because of their successful implementation, the approach seemed promising. The use of Genetic Programming is also supported by Colledanchise and Ögren (2018, pp. 139–149), who suggest the learning method as well in similar contexts. Lieto (2021, p. 33) notes that those learning algorithms are known for working well if the problem space is known little. In contrast, Russell and Norvig (2021, pp. 21–22) point out the danger of an explosion of possible combinations by using Genetic Programming. This comes along with long computing times, which we could observe in our examples. Since the available

time resources are constrained by the scope of a master's thesis, other learning approaches, further implementations and longer learning intervals in the number of weeks were not pursued.

After we defined BTs and gave an introduction to the terminology, possible adaptions, and different learning approaches in Chapter 2, we applied the method to the benchmark example Pac-Man to make first experiences within a controlled and comparable setting in Chapter 3. By implementing the Pac-Man example, we observed several aspects and gathered insights, which we used partly for the implementation of the forklift truck. As one of the lessons learned from the Pac-Man example, we point out that the formulation of the fitness function is crucial. Because within this function the goal needs to be translated into a mathematical expression, there are at least two critical aspects. The former is that the goal or the goals and subgoals need to be quantitative or Boolean expressible. This works well, by using the score of the game as the overall goal. Other subgoals, like not to be caught by the ghosts are inherent in this goal formulation, but not explicit. This leads to the latter aspect, which can be seen as the preciousness of the mathematical expression. Because the goal needs to be translated, a process of abstraction is conducted, which we also analysed in Chapter 5 in connection with the cognitive functions. Through the abstraction process, there is a loss of information, which entails the risk of describing the goal preciously enough for a satisfactory learning result. In this regard, we note that the formulation of the fitness function can be error sensitive.

Using the Pac-Man example, we were able to show that the Genetic Programming algorithm generates BTs that achieve a similar score on average to the hand-made ones. We observed that the learned BTs can cause unpredictable behaviour, due to the combination of the different nodes and their structure. This shows that similar BTs, regarding the score, can be learned, but with the cost of potentially unintended behaviour, which becomes crucial in the context of safety guarantees.

During the implementation and optimisation process of the learning algorithm for the forklift truck, we observed that the granularity of the execution nodes needs to be adjusted to the complexity of the task to learn. We have started with the task of loading one pallet. In the second step, we could show that the AI system can integrate a repeater node to load and unload two pallets and test an additional condition. So we can deduce that scalability to more pallets is possible. We like to

mention that the learned BTs were intended by us. Through that, we were able to adjust the parameters of the learning algorithm to get suitable BTs. So we can deduce that the designer needs to have a rough idea of how the BT should look like. We also observed that the granularity of the execution nodes should fit the level of abstraction of the simulation, which is used for the learning algorithm. This matching can differ only in one direction. It is possible to use coarse-grained execution nodes in combination with a highly detailed simulation, like a physical one in our case. The consequence is long computing times. But this does not work the other way around. It is not possible to use a simulation on a high abstract level, e.g. a simple Finite State Machine while having fine-grained execution nodes[23]. This is because the states in between the different actions cannot be represented in an abstract simulation. Because the fitness function is connected with the simulation through e.g. reward specific external states in the environment, the expressional power of the fitness function is limited as well. From that, we deduce that for learning more complex scenarios with this approach of a Genetic Programming algorithm, the granularity of the execution nodes needs to be more fine-grained than the ones used in our example.

An example of a more complex scenario could be that one pallet is blocked by another, so the blocking pallet needs to be taken away at first or the desired pallet is approached from the backside. We did not examine such complex scenarios with our approach, since finer-grained execution nodes cause more possible combinations with the need for a larger BT. In the Pac-Man example, we learned larger BTs, as shown in Appendix C, but this comes with higher computational costs. The simulation of the Pac-Man game was about 20 times (2 seconds vs.. 40 seconds) faster than the one of the Crayler for two pallets. By comparing the level of complexity of the task environment in Table 6, it is visible that the Pac-Man environment and the Crayler simulation are simpler than a real-world scenario. We like to note that BTs as such are developed in the game industry, in which mostly domain-specific or rather simple task environments occur. However, they do not cover the complexity of the real world. In this sense, it could be possible that our

[23] We assume that it is practically not possible through the endless number of possible configurations of nodes since the size of the BT is unrestricted and only controlled by the cost. We like to admit that there is a theoretical possibility that the learning algorithm finds a solution by accident. However, we do not categorise such an approach as practical, since the proceeding is unguided, because of the lack of subgoals.

approach, to learning BTs with Genetic Programming, is limited. The boundary is set through the increasing number of possible combinations of execution nodes and their structure, which are necessary to learn more complex tasks.
A possible solution to push this boundary further could lay in the property of the BTs itself to be hierarchical and modular. Through that, only parts of a BT can be learned or hybrid forms can be integrated, which are more adaptive to different environments due to the use of emergent methods in a subtree. We assume that the security guarantees can be ensured by manually embedding these sub-trees in the general BT structure by restricting and continuously checking security-relevant conditions. In this regard, we like to mention the concern that there is the possibility that not all safety-relevant situations, which could occur in the real world can be covered by a structure of conditions.

6.2 Summary and Discussion of the Cognitive Investigation

In the second part of this work, we focused on the question of which cognitive functions are inherent within the implemented AI system of the forklift truck. By identifying these cognitive functions, we could in consequence detect possible aspects of improvement at those, which are associated on the side of the designer. To do so, we categorised the task from a theoretical perspective. We analysed that we deal with a problem-solving task, which can be described and rationalised within a problem space. For the further process, we focused the investigation within Marr's Levels of Analysis on the level of computational theory, through which, the comparison of the cognitive functions was set on a descriptive and functional level. Furthermore, we gave an introduction to cognitive architectures and described the LRMB as a model of the human brain. We like to admit that this model is rather unknown, compared to other cognitive architectures mentioned in the survey of Kotseruba and Tsotsos (2020), like SOAR or ACT-R. In this sense, its reliability and validity can be questioned, not least because there is no other literature evaluating the model to our knowledge. We chose it because it fits our purposes due to its descriptive structure and comparability. According to Wang et al. (2006), the different cognitive functions are related to each other. This opens up the question of whether single cognitive functions are distinguishable, and if so, where are the

boundaries between them? On the other side, we value the descriptive power of such a model and assume that the access through a qualitative comparison is fruitful.

It is debatable if our assumption holds that the problem-solving of humans, based on the LRMB is comparable with the problem-solving of a machine. There is the possibility that there exists such a major difference within the process and the structure of problem-solving between machines and humans that a comparison on the level of cognitive functions is not effective. An argument for that is the distinction between a weak and a strong AI. If consciousness or emotional states, which we explicitly excluded, are substantial for human problem-solving, our approach is questionable. More generally, Lieto et al. (2018) state that one of the main roles of cognitive architectures, which try to model human cognition, is to develop AIs by investigating the underlying mechanisms and functions. Since problem-solving is a part of human cognition, this provides an argument that our assumption holds.

We identified that it was necessary to define the notion of “knowledge” in the context of the AI system. By doing so, we approached the term from different perspectives and categorized various kinds of knowledge through distinctions. Through that, we gathered the first insights with regard to the analysis of the cognitive functions and their allocation to the AI system or the designer. We do not hold the truth for a definition of “knowledge” nor did we consider all possible definitions, so that our investigation can only be seen as an approach. Nevertheless, it showed the relation between the term “knowledge” and representational structures, which became visible with Newel’s concept of a Knowledge Level. We also showed that “knowledge” can be accessed through the paradigm of 4E cognition, but we did not include these insights in our suggestions for improvement. Here we see potential for further investigations and improvement, e.g. by integrating force sensors for precious and direct control, at those parts, where the visual sensors are not reliable enough.

When analysing the cognitive functions and assigning them to the three categories in Figure 34, it is possible to criticize whether the AI system is "fully capable" of even just one of these functions. As we stated, the AI system mostly needs some kind of structure to conduct a cognitive function. For example, there is no object identification without inserting which objects should be identified. This includes setting some kind of goal, which can be explicit, like in a fitness function or implicit,

e.g. providing a structure or labelled training data. The point is that the designer must incorporate some kind of "knowledge" into the machine in some form. Here we face the previously mentioned critique of where to put the boundary of a cognitive function. Only in passing, this opens up the discussion, if an AI can just be a constructed Weak AI, made by humans, or expressed with Newel's words, we are just able to construct a homunculus but no "real intelligence".

6.3 Outlook and Further Suggestions

At the end of Chapter 5, we have defined the notion of an "intelligent agent", summarised the analysis of the cognitive functions and based on that made suggestions for further improvements, which integrate an "artificial abstraction" through "subgoaling" and comprehension through the embedding of learned subtrees. We pointed out that besides safety guarantees and human readability, the adaption to different environments and situations is crucial for an intelligent behaving forklift truck. Similar to the trend of becoming hybrid models, which we observed in cognitive architectures like ACT-R and SOAR, we suggested including emergent elements in the AI system. We argued that BTs are well-suitable to provide a framework for a hybrid control structure due to their hierarchical and modular properties. To include emergent elements in a safe setting, we suggest embedding them as subtrees, which are only executed under safe conditions and continuously checked. Through that, the forklift truck could adapt to a wider range of different environments, since this is the strength of emergent techniques. The weaknesses, like being unpredictable could be contained by the BT structure. A similar approach with *Artificial Neural Networks* is suggested by Sprague and Ögren (2022). At this point, we see a potential for further research by analysing missing cognitive functions within the AI system to identify suitable emergent techniques, which can be included in a BT structure as subtrees.

As a further outlook, our approach of analysing AI through cognitive functions could potentially contribute to understanding machine behaviour. The necessity of investigating the field is shown by a review by Rahwan et al. (2019), which is published in the journal "Nature". By analysing the cognitive functions of an AI, it is possible to investigate what the system is and is not capable of on a descriptive

level. This could help to categorise and predict the boundaries of a machine's behaviour and could contribute to better comprehending and improving AI systems.

7 Conclusion

This research aims to develop a possible learnable agent program for autonomous utility machinery on the concrete example of the forklift truck, based on BTs, and to make further suggestions towards an intelligent agent by investigating the system according to its cognitive functions. The results stay in the light of a human-understandable AI, which provides the precondition for safety guarantees in a real-world application. The central research question has a practical-oriented first part and a theoretical-oriented second part:

RQ: What does a learnable agent program based on the framework of Behaviour Trees for an autonomous forklift truck in the concrete scenario of the Crayler look like, and which cognitive functions are additionally required towards intelligent behaviour?

To answer the first part of the research question, we implemented a learning framework, which uses Genetic Programming to learn BTs. After testing the method with the Benchmark example Pac-Man, we applied it to the forklift truck. We could show that Genetic Programming provides a suitable method to learn BTs for our specific task environment in a physical simulation. The implemented AI system as a whole, including the learning framework, provides a learnable agent program for the forklift truck and answers the first part of the research question. However, we observed that the designer needs to be engaged in the learning process regarding the specification of the fitness function and the matching of the granularity of execution nodes with the level of abstraction of the simulation to adapt to the complexity of the task to be learnt. The more the designer intervenes, the less autonomous the AI system becomes.
This gives rise to investigating the AI system from a cognitive perspective. Through the comparison of cognitive functions, which are involved in (human) problem-solving, we analysed that the AI system is not able to “abstract” nor to “comprehend”, which answers the second part of the research question. Based on these findings, we are able to make further suggestions towards an intelligent agent. This includes the embedding of learned subtrees in larger BT structure to

ensure “comprehension” and the automatization of the learning process with Genetic Programming, which we called “artificial abstraction”. Inspired by a trend in cognitive architectures, we argue for using BTs as a framework for hybrid AI that is potentially understandable to humans through the integration of specified nodes, using emergent methods, in a controlled structure.

Our findings have a bidirectional impact. From the practical perspective, the made suggestions can guide the research of the forklift truck at the AIT towards an increasingly autonomous and intelligent utility vehicle. From a theoretical perspective, the analysis through cognitive functions can be applied to other AI systems to get a better understanding of their machine behaviour. Using BTs as a framework for hybrid AI in a potentially safe setting can inspire further research in the field.

8 References

Abrahamsen, A., & Bechtel, W. (2012). History and core themes. In K. Frankish & W. Ramsey (Eds.), *The Cambridge Handbook of Cognitive Science* (pp. 9–28). Cambridge University Press. https://doi.org/10.1017/CBO9781139033916

Anderson, J. R., Bothell, D., Byrne, M. D., Douglass, S., Lebiere, C., & Qin, Y. (2004). An integrated theory of the mind. *Psychological Review*, *111*(4), 1036–1060.

Andrews, A., Thesden, L., Lönnhager, D., Jobson, T., Dudfield, R., Dudfield, N., & Kluyver, T. (2023). *Pygame* (Version 2.3.0) [Computer software]. https://www.pygame.org/news

Andries, M., Chavez-Garcia, R. O., Chatila, R., Giusti, A., & Gambardella, L. M. (2018). Affordance Equivalences in Robotics: A Formalism. *Frontiers in Neurorobotics*, *12*. https://www.frontiersin.org/articles/10.3389/fnbot.2018.00026

Asselman, A., Aammou, S., & Nasseh, A.-E. (2015). Comparative study of cognitive architectures. *International Research Journal of Computer Science (IRJCS)*, *2*(9), 8–13.

Banerjee, B. (2018). Autonomous Acquisition of Behavior Trees for Robot Control. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3460–3467. https://doi.org/10.1109/IROS.2018.8594083

Barros, R. C., de Carvalho, A. C. P. L. F., & Freitas, A. A. (2015). *Automatic Design of Decision-Tree Induction Algorithms* (1st ed. 2015). Springer International Publishing : Imprint: Springer. https://doi.org/10.1007/978-3-319-14231-9

Box, G. E. P. (1976). Science and Statistics. *Journal of the American Statistical Association*, *71*(356), 791–799. https://doi.org/10.1080/01621459.1976.10480949

Boy, G. A. (1998). Cognitive function analysis for human-centered automation of safety-critical systems. In Proceedings of the SIGCHI conference on Human factors in computing systems, 265–272.

Brendel, E., & Gähde, U. (2016). Was ist Wissen? In W. Buchmüller & C. Jakobeit (Eds.), *Erkenntnis, Wissenschaft und Gesellschaft: Wie Forschung Wissen schafft* (pp. 9–21). Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-662-49912-2_2

Brochu, E., Cora, V. M., & De Freitas, N. (2010). A tutorial on Bayesian optimization of expensive cost functions, with application to active user

modeling and hierarchical reinforcement learning. *arXiv Preprint arXiv:1012.2599*.

Brooks, R. A. (1990). Elephants don't play chess. *Robotics and Autonomous Systems*, *6*(1), 3–15. https://doi.org/10.1016/S0921-8890(05)80025-9

Brooks, R. A. (1991). Intelligence without representation. *Artificial Intelligence*, *47*(1), 139–159. https://doi.org/10.1016/0004-3702(91)90053-M

Champandard, A. J., & Dunstan, P. (2019). The Behavior Tree Starter Kit. In *Game AI Pro 360. Guide to architecture* (1st ed.). CRC Press.

Chartrand, G., Zhang, P., & Lesniak, L. (2015). *Graphs and digraphs* (Sixth edition.). Chapman and Hall/CRC.

Clark, A. (2012). Embodied, embedded, and extended cognition. In K. Frankish & W. Ramsey (Eds.), *The Cambridge Handbook of Cognitive Science* (pp. 275–291). Cambridge University Press. https://doi.org/10.1017/CBO9781139033916

Clark, A., & Chalmers, D. (1998). The Extended Mind. *Analysis*, *58*(1), 7–19. http://www.jstor.org/stable/3328150

Colledanchise, M., & Ögren, P. (2016). How Behavior Trees generalize the Teleo-Reactive paradigm and And-Or-Trees. *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 424–429. https://doi.org/10.1109/IROS.2016.7759089

Colledanchise, M., & Ögren, P. (2018). *Behavior Trees in Robotics and AI: An Introduction: Vol. abs/1709.00084* (1st Edition). CRC Press. http://arxiv.org/abs/1709.00084

Colledanchise, M., Parasuraman, R., & Ögren, P. (2019). Learning of Behavior Trees for Autonomous Agents. *IEEE Transactions on Games*, *11*(2), 183–189. https://doi.org/10.1109/TG.2018.2816806

Collins, S., Ruina, A., Tedrake, R., & Wisse, M. (2005). Efficient Bipedal Robots Based on Passive-Dynamic Walkers. *Science (American Association for the Advancement of Science)*, *307*(5712), 1082–1085.

David Marr. (2010). The Philosophy and the Approach. In *Vision: A Computational Investigation Into the Human Representation and Processing of Visual Information* (pp. 8–38). The MIT Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=324708&site=ehost-live

Davis, R., Shrobe, H., & Szolovits, P. (1993). What is a knowledge representation? *AI Magazine*, *14*(1), 17–17.

Dey, R., & Child, C. (2013). QL-BT: Enhancing behaviour tree design and implementation with Q-learning. *2013 IEEE Conference on Computational*

Inteligence in Games (CIG), 1–8. https://doi.org/10.1109/CIG.2013.6633623

Escudero, D. (2022). *Learning Behavior Trees using Genetic Programming* [Computer software]. Retrieved May 30, 2023, from https://github.com/dgerod/behavior_tree_learning

FFG - Die Österreichische Forschungsförderungsgesellschaft. (2019, October 1). *HOPPER Handling of man-made objects using automated positioning, planning and enhanced reasoning methods*. Retrieved April 18, 2023, from https://projekte.ffg.at/projekt/3321101

Florez-Puga, G., Gomez-Martin, M. A., Gomez-Martin, P. P., Diaz-Agudo, B., & Gonzalez-Calero, P. A. (2009). Query-Enabled Behavior Trees. *IEEE Transactions on Computational Intelligence and AI in Games*, *1*(4), 298–308. https://doi.org/10.1109/TCIAIG.2009.2036369

Friston, K. (2010). The free-energy principle: A unified brain theory? *Nature Reviews Neuroscience*, *11*(2), 127–138. https://doi.org/10.1038/nrn2787

Fu, Y., Qin, L., & Yin, Q. (2016). A Reinforcement Learning Behavior Tree Framework for Game AI. *Proceedings of the 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering*. 2016 International Conference on Economics, Social Science, Arts, Education and Management Engineering, Huhhot, China. https://doi.org/10.2991/essaeme-16.2016.120

Gallagher, S. (2017). *Enactivist Interventions: Rethinking the Mind* (1st ed.). Oxford University Press.

Galvan-Lopez, E., Fagan, D., Murphy, E., Swafford, J. M., Agapitos, A., O'Neill, M., & Brabazon, A. (2010). Comparing the performance of the evolvable pi-Grammatical Evolution genotype-phenotype map to Grammatical Evolution in the dynamic Ms. Pac-Man environment. *IEEE Congress on Evolutionary Computation*, 1–8. https://doi.org/10.1109/CEC.2010.5586508

Gettier, E. L. (1963). Is Justified True Belief Knowledge? *Analysis*, *23*(6), 121–123. https://doi.org/10.1093/analys/23.6.121

Ghzouli, R., Berger, T., Johnsen, E. B., Dragule, S., & Wąsowski, A. (2020). Behavior trees in action: A study of robotics applications. *Proceedings of the 13th ACM SIGPLAN International Conference on Software Language Engineering*, 196–209. https://doi.org/10.1145/3426425.3426942

Goerz, R. (2023). *The Memory of Attractors in Recurrent Neuronal Networks and its Application to Human-Machine Interaction*. *16*. Retrieved April 18, 2023, from https://journals.phl.univie.ac.at/meicogsci/article/view/442

Goerz, R., Horvat, M. M., & Simonič, M. (2023). *PiH Exception strategies Multimodal Database* [Dataset]. Retrieved April 18, 2023, from https://doi.org/10.5281/ZENODO.7568592

Goerz, R., Simonič, M., Ude, A., & Horvat, M. M. (2023). *Trajectory-Based Exception Learning Strategy in the Context of Robotics* (A. Žemva & A. Trost, Eds.). Društvo Slovenska sekcija IEEE. https://erk.fe.uni-lj.si/2023/papers/goerz(trajectory_based_exception).pdf

Gugliermo, S., Schaffernicht, E., Koniaris, C., & Pecora, F. (2023). Learning Behavior Trees from Planning Experts Using Decision Tree and Logic Factorization. *IEEE Robotics and Automation Letters*, 1–8. IEEE Robotics and Automation Letters. https://doi.org/10.1109/LRA.2023.3268598

Hannaford, B., Hu, D., Zhang, D., & Li, Y. (2016). *Simulation Results on Selector Adaptation in Behavior Trees* (arXiv:1606.09219). arXiv. http://arxiv.org/abs/1606.09219

Harre, R. (2006). Gilbert Ryle—The Concept of Mind. In J. Shand, *Central Works of Philosophy V4: Twentieth Century: Moore to Popper* (Issue Volume 4, The twentieth century, pp. 214–238). Routledge. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=944120&site=ehost-live

Herbert A. Simon. (2019). *The Sciences of the Artificial: Vol. Third edition [2019 edition]*. The MIT Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=2204640&site=ehost-live

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., Melo, G. D., Gutierrez, C., Kirrane, S., Gayo, J. E. L., Navigli, R., & Neumaier, S. (2021). Knowledge graphs. *ACM Computing Surveys (Csur)*, *54*(4), 1–37.

HOPPER - AIT Austrian Institute Of Technology. (2022, September 30). ait.ac.at. Retrieved December 13, 2023, from https://www.ait.ac.at/ueber-das-ait/center/center-for-vision-automation-control/complex-dynamical-systems/projekte/hopper?no_cache=1

Ichikawa, J. J., & Steup, M. (2018). The Analysis of Knowledge. In *The Stanford Encyclopedia of Philosophy* (Edward N. Zalta (ed.)). Metaphysics Research Lab, Stanford University. https://plato.stanford.edu/archives/sum2018/entries/knowledge-analysis/

Iovino, M., Scukins, E., Styrud, J., Ögren, P., & Smith, C. (2022). A survey of Behavior Trees in robotics and AI. *Robotics and Autonomous Systems*, *154*, 104096. https://doi.org/10.1016/j.robot.2022.104096

Iovino, M., Styrud, J., Falco, P., & Smith, C. (2021). Learning Behavior Trees with Genetic Programming in Unpredictable Environments. *2021 IEEE International Conference on Robotics and Automation (ICRA)*, 4591–4597. https://doi.org/10.1109/ICRA48506.2021.9562088

Kaufmann, M., Wagner, D., & van Leeuwen, J. (2003). *Drawing Graphs: Methods and Models* (Vol. 2025). https://doi.org/10.1007/3-540-44969-8

Kiely, K. M. (2014). Cognitive Function. In A. C. Michalos (Ed.), *Encyclopedia of Quality of Life and Well-Being Research* (pp. 974–978). Springer Netherlands. https://doi.org/10.1007/978-94-007-0753-5_426

Kotseruba, I., & Tsotsos, J. K. (2020). 40 years of cognitive architectures: Core cognitive abilities and practical applications. *Artificial Intelligence Review*, *53*(1), 17–94. https://doi.org/10.1007/s10462-018-9646-y

Koza, JohnR. (1994). Genetic programming as a means for programming computers by natural selection. *Statistics and Computing*, *4*(2). https://doi.org/10.1007/BF00175355

Kumar, R., & Pattnaik, P. K. (2018). *Graph Theory*. Laxmi Publications Pvt Ltd. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=2228702&site=ehost-live

Laird, J. E., Lebiere, C., & Rosenbloom, P. S. (2017). A standard model of the mind: Toward a common computational framework across artificial intelligence, cognitive science, neuroscience, and robotics. *Ai Magazine*, *38*(4), 13–26.

Leamy, D. (2020). *Pacman* [Computer software]. Retrieved May 16, 2023, from https://github.com/DevinLeamy/Pacman/tree/master/Pacman

Legg, S., & Hutter, M. (2007). A collection of definitions of intelligence. *Frontiers in Artificial Intelligence and Applications*, *157*, 17.

Lieto, A. (2021). *Cognitive design for artificial minds*. Routledge,.

Lieto, A., Bhatt, M., Oltramari, A., & Vernon, D. (2018). The role of cognitive architectures in general artificial intelligence. *Cognitive Systems Research*, *48*, 1–3.

Lieto, A., Lebiere, C., & Oltramari, A. (2018). The knowledge level in cognitive architectures: Current limitations and possible developments. *Cognitive Architectures for Artificial Minds*, *48*, 39–55. https://doi.org/10.1016/j.cogsys.2017.05.001

Lowe, E. J. (2005). *A priori and a posteriori*. Oxford University Press. https://doi.org/10.1093/acref/9780199264797.013.0123

Martin, E., & Hine, R. (2015). *Behaviour*. Oxford University Press. https://doi.org/10.1093/acref/9780198714378.013.0455

Merrill, B. (2019). Building Utility Decisions into Your Existing Behavior Tree. In *Game AI Pro 360: Guide to Architecture* (1st ed., pp. 81–90). CRC Press.

Miłkowski, M. (2018). Embodied Cognition. In M. Sprevak & M. Colombo (Eds.), *The Routledge Handbook of the Computational Mind* (1st ed., pp. 323–338). Routledge. https://doi.org/10.4324/9781315643670

Newell, A. (1974). You can't play 20 questions with nature and win. *Visual Information Processing*.

Newell, A. (1979). *Reasoning, problem solving and decision processes: The problem space as a fundamental category*. *8*.

Newell, A. (1982). The knowledge level. *Artificial Intelligence*, *18*(1), 87–127.

Newell, A. (1994). *Unified theories of cognition*. Harvard University Press.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. *Commun. ACM*, *19*(3), 113–126. https://doi.org/10.1145/360018.360022

Newen, A., De Bruin, L., & Gallagher, S. (2018). *The Oxford handbook of 4E cognition*. Oxford University Press.

Nicolau, M., Perez-Liebana, D., O'Neill, M., & Brabazon, A. (2017). Evolutionary Behavior Tree Approaches for Navigating Platform Games. *IEEE Transactions on Computational Intelligence and AI in Games*, *9*(3), 227–238. IEEE Transactions on Computational Intelligence and AI in Games. https://doi.org/10.1109/TCIAIG.2016.2543661

Nilsson, N. (1993). *Teleo-Reactive Programs for Agent Control* (arXiv:cs/9401101). arXiv. https://doi.org/10.48550/arXiv.cs/9401101

Nogueira, F. (2014). *Bayesian Optimization* [Computer software]. Retrieved January 12, 2024, from https://github.com/fmfn/BayesianOptimization

Ocio, S. (2012). Adapting AI Behaviors to Players in Driver San Francisco Hinted-Execution Behavior Trees. *Proceedings of the Eighth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 51–56.

Oxford English Dictionary. (2023). *Inference, n.* Oxford University Press. https://doi.org/10.1093/OED/3991017993

Palfinger AG. (2022, November). *Palfinger Crayler BM 214*. Retrieved December 14, 2023, from https://assets.palfinger.com/importdata/product-data/truck-mounted-forklifts/brochures/BM%20Serie/KP-TMFM1%2BEN_Ansicht%20Boxm..pdf

Pereira, R. de P., & Engel, P. M. (2015). *A Framework for Constrained and Adaptive Behavior-Based Agents* (arXiv:1506.02312). arXiv. http://arxiv.org/abs/1506.02312

Peschl, M. (2021, October 25). *Cognitive Science—Introduction and Basic Concepts* [Lecture].

Pezzato, C., Corbato, C. H., Bonhof, S., & Wisse, M. (2022). Active Inference and Behavior Trees for Reactive Action Planning and Execution in Robotics. *IEEE Transactions on Robotics*, 1–20. https://doi.org/10.1109/TRO.2022.3226144

Poli, R., Langdon, W. B., McPhee, N. F., & Koza, J. R. (2008). *A field guide to genetic programming*. Lulu Press].

Rahwan, I., Cebrian, M., Obradovich, N., Bongard, J., Bonnefon, J.-F., Breazeal, C., Crandall, J. W., Christakis, N. A., Couzin, I. D., Jackson, M. O., Jennings, N. R., Kamar, E., Kloumann, I. M., Larochelle, H., Lazer, D., McElreath, R., Mislove, A., Parkes, D. C., Pentland, A. 'Sandy', … Wellman, M. (2019). Machine behaviour. *Nature*, *568*(7753), 477–486. https://doi.org/10.1038/s41586-019-1138-y

Rasmussen, C. E., & Williams, C. K. I. (2008). *Gaussian processes for machine learning* (3. print). MIT Press.

Robertson, G., & Watson, I. (2015). Building behavior trees from observations in real-time strategy games. *2015 International Symposium on Innovations in Intelligent SysTems and Applications (INISTA)*, 1–7. https://doi.org/10.1109/INISTA.2015.7276774

Roland, J. (1958). On 'Knowing How' and 'Knowing That'. *The Philosophical Review*, *67*(3), 379–388. https://doi.org/10.2307/2182398

Rovida, F., Wuthier, D., Grossmann, B., Fumagalli, M., & Krüger, V. (2018). Motion Generators Combined with Behavior Trees: A Novel Approach to Skill Modelling. *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 5964–5971. https://doi.org/10.1109/IROS.2018.8594319

Rowlands, M. (2006). *Body language: Representation in action*. MIT Press,.

Roy, E. (2013). Cognitive Function. In M. D. Gellman & J. R. Turner (Eds.), *Encyclopedia of Behavioral Medicine* (pp. 448–449). Springer New York. https://doi.org/10.1007/978-1-4419-1005-9_1117

Russell, S. J., & Norvig, P. (2021). *Artificial intelligence: A modern approach* (Fourth edition.). Pearson.

Ryle, G. (2009). *The Concept of Mind: 60th Anniversary Edition* (1st ed.). Routledge. https://doi.org/10.4324/9780203875858

Salisbury, J., & Schneider, S. (2018). Concepts, Symbols, and Computation. In M. Sprevak & M. Colombo (Eds.), *The Routledge Handbook of the Computational Mind* (1st ed., pp. 310–322). Routledge. https://doi.org/10.4324/9781315643670

Searle, J. R. (1980). Minds, brains, and programs. *Behavioral and Brain Sciences*, *3*(3), 417–424.

Sims, K. (1994). Evolving 3D Morphology and Behavior by Competition. In R. Allen. Brooks, P. Maes, R. A. Brooks, P. Maes, & International Workshop on the Synthesis and Simulation of Living Systems (Eds.), *Artificial life IV: proceedings of the Fourth International Workshop on the Synthesis and Simulation of Living Systems* (pp. 28–39). MIT Press,. http://search.ebscohost.com/login.aspx?direct=true&scope=site&db=nlebk&AN=1707

Snoek, J., Larochelle, H., & Adams, R. P. (2012). Practical bayesian optimization of machine learning algorithms. *Advances in Neural Information Processing Systems*, *25*.

Sorensen, M. L. S., & Rebay-Salisbury, K. (2013). *Embodied Knowledge: Historical Perspectives on Belief and Technology: Vol. 1st ed*. Oxbow Books. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=551974&site=ehost-live

Sprague, C. I., & Ögren, P. (2022). Adding Neural Network Controllers to Behavior Trees without Destroying Performance Guarantees. *2022 IEEE 61st Conference on Decision and Control (CDC)*, 3989–3996. https://doi.org/10.1109/CDC51059.2022.9992501

Stonier, D. (2020, August 11). *Py Trees Documentation*. Retrieved May 21, 2023, from https://py-trees.readthedocs.io/en/release-2.0.x/index.html#index-section

Stonier, D. (2023). *Py_trees* (Version 2.2.3) [Computer software]. May 21, 2023, from https://github.com/splintered-reality/py_trees

Sutton, R. S., Precup, D., & Singh, S. (1999). Between MDPs and semi-MDPs: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, *112*(1–2), 181–211. https://doi.org/10.1016/S0004-3702(99)00052-1

Thomas Back. (1996). *Evolutionary Algorithms in Theory and Practice: Evolution Strategies, Evolutionary Programming, Genetic Algorithms*. Oxford University Press. https://search.ebscohost.com/login.aspx?direct=true&db=nlebk&AN=287536&site=ehost-live

Turing, A. M. (1950). Mind. *Mind*, *59*(236), 433–460.

Uher, J. (2016). What is Behaviour? And (when) is Language Behaviour? A Metatheoretical Definition. *Journal for the Theory of Social Behaviour*, *46*(4), 475–501. https://doi.org/10.1111/jtsb.12104

Wang, G., Yang, J., & Xu, J. (2017). Granular computing: From granularity optimization to multi-granularity joint problem solving. *Granular Computing*, *2*, 105–120.

Wang, Y. (2010). Cognitive robots: A reference model toward intelligent authentication. *IEEE Robotics & Automation Magazine*, *17*(4), 54–62.

Wang, Y., & Chiew, V. (2010). On the cognitive process of human problem solving. *Cognitive Systems Research*, *11*(1), 81–92.

Wang, Y., Wang, Y., Patel, S., & Patel, D. (2006). A layered reference model of the brain (LRMB). *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *36*(2), 124–133.

Wei, J., & Salvendy, G. (2004). The cognitive task analysis methods for job and task design: Review and reappraisal. *Behaviour & Information Technology*, *23*(4), 273–299.

Zhang, Q., Xu, K., Jiao, P., & Yin, Q. (2018). Behavior Modeling for Autonomous Agents Based on Modified Evolving Behavior Trees. *2018 IEEE 7th Data Driven Control and Learning Systems Conference (DDCLS)*, 1140–1145. https://doi.org/10.1109/DDCLS.2018.8515939

Zhang, Q., Yao, J., Yin, Q., & Zha, Y. (2018). Learning Behavior Trees for Autonomous Agents with Hybrid Constraints Evolution. *Applied Sciences*, *8*(7), Article 7. https://doi.org/10.3390/app8071077

9 Appendix

Appendix A Final Parameters of the Genetic Programming Algorithm for the Example of Pac-Man

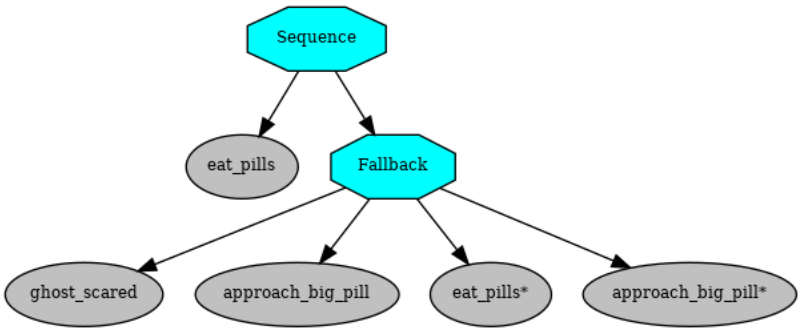
```
########################################
# Parameters for Genetic Programming
########################################

#Population and Generations:
gen_parameters.ind_start_length: int = 8                              # Start length of initial genomes
gen_parameters.min_length: int = 4                                    # Minimum length of individual
gen_parameters.n_population: int = 16                                 # Number of individuals in population
gen_parameters.n_generations: int = 1000                              # Maximum number of generations (if no one fails or
reaches                                                                 the threshold before) -> defined in a list of several
                                                                        runs
generatons_run_in_row = [100 , 200 , 500 , 1000]                     # List of max generations by one trail

#Mutation, Crossover and Selection:
gen_parameters.f_crossover: float = 0.5                               # Fraction of parent pool selected for crossover
gen_parameters.n_offspring_crossover: int = 2                         # Number of offspring from crossover per parent
gen_parameters.replace_crossover: bool = True                         # Crossover replaces subtree at receiving genome or
inserts
gen_parameters.f_mutation: float = 0.7                                # Fraction of parent pool selected for mutation
gen_parameters.n_offspring_mutation: int = 2                          # Number of offspring from mutation per parent
gen_parameters.parent_selection: int = GeneticSelectionMethods.RANK   # Selection method for parents
gen_parameters.survivor_selection: int = GeneticSelectionMethods.RANK # Selection method for survival
gen_parameters.f_elites: float = 0.1                                  # Fraction of population that survive as elites
gen_parameters.f_parents: float = gen_parameters.f_elites             # Fraction of parents that may survive to next generation
gen_parameters.mutate_co_offspring: bool = False                      # Offspring from crossover may also be mutated
gen_parameters.mutate_co_parents: bool = True                         # Parents for crossover may also be mutated
gen_parameters.mutation_p_add: float = 0.2                            # Probability of mutation adding a gene
gen_parameters.mutation_p_delete: float = 0.6                         # Probability of mutation deleting a gene
gen_parameters.allow_identical: bool = True                           # Offspring may be identical to any parent in prev
                                                                        generation

#Baseline:
gen_parameters.keep_baseline: bool = True                             # Baseline, if any, is always kept in population for
breeding
gen_parameters.boost_baseline: bool = False                           # Baseline is boosted to have higher probability of
breeding
gen_parameters.boost_baseline_only_co: bool = True                    # Baseline is boosted for crossover selection, not
mutation



#Fitness and Hash:
gen_parameters.fitness_threshold: float = 100000.0                    # Finish when best fitness is over this threshold
gen_parameters.hash_table_size: int = 100000                          # Size of hash table
gen_parameters.rerun_fitness: int = 5                                 # 0-run only once, 1-according to prob, 2-always
                                                                      # Gets fitness from hash table if possible, otherwise
from                                                                    simulation
                                                                    #rerun = 0 means never rerun
                                                                    #rerun = 1 means rerun with diminishing probability
                                                                    #rerun = 2 means rerun always
                                                                    #rerun = n with n>=3 means 3 reruns always
```

Appendix B Specifications of the Used Computer

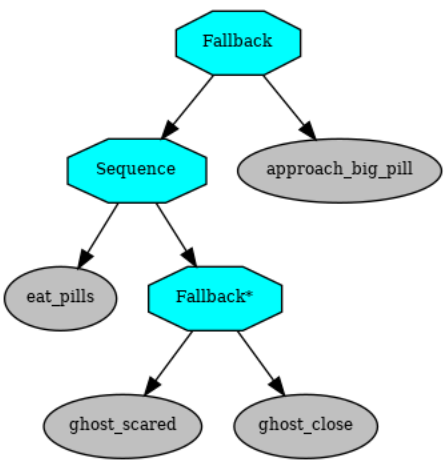
	PC	**Server**
Processor	11th Gen Intel Core i7 3,00GHz	Intel Xeon W-2145CPU 3.70GHz
RAM	32,0 GB	32,0 GB
System	Windows 10 Enterprise	Windows 10 Enterprise
Systemtype	64-Bit	64-Bit

Appendix C Best Individuals of the Latest Iteration Step of the Pac-Man Benchmark

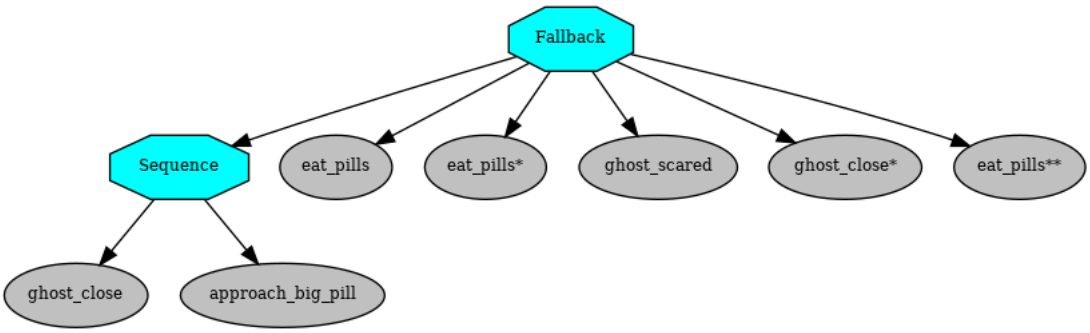
- 100 Generations:



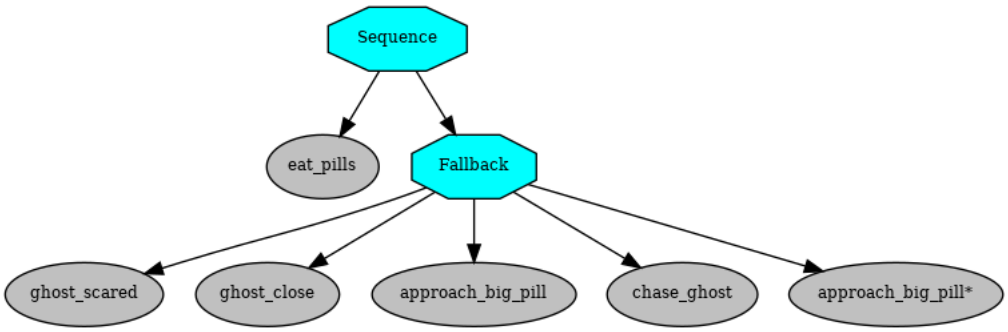
- 200 Generations:



- 500 Generations:



- 1000 Generations:



Appendix D Final Parameters of the Genetic Programming Algorithm for the Crayler Simulation for One Pallet

```
#######################################
# Parameters for Genetic Programming
#######################################

#Population and Generations:
gen_parameters.ind_start_length: int = 8                          # Start length of initial genomes
gen_parameters.min_length: int = 2                                # Minimum length of individual
gen_parameters.n_population: int = 8                              # Number of individuals in population
gen_parameters.n_generations: int = 200                           # Maximum number of generations

#Mutation, Crossover and Selection:
gen_parameters.f_crossover: float = 0.5                           # Fraction of parent pool selected for crossover
gen_parameters.n_offspring_crossover: int = 2                     # Number of offspring from crossover per parent
gen_parameters.replace_crossover: bool = True                     # Crossover replaces subtree at receiving genome or
inserts
gen_parameters.f_mutation: float = 0.7                            # Fraction of parent pool selected for mutation
gen_parameters.n_offspring_mutation: int = 2                      # Number of offspring from mutation per parent
gen_parameters.parent_selection: int = GeneticSelectionMethods.RANK  # Selection method for parents
gen_parameters.survivor_selection: int = GeneticSelectionMethods.RANK # Selection method for survival
gen_parameters.f_elites: float = 0.1                              # Fraction of population that survive as elites
gen_parameters.f_parents: float = gen_parameters.f_elites         # Fraction of parents that may survive to next generation
gen_parameters.mutate_co_offspring: bool = False                  # Offspring from crossover may also be mutated
gen_parameters.mutate_co_parents: bool = True                     # Parents for crossover may also be mutated
gen_parameters.mutation_p_add: float = 0.2                        # Probability of mutation adding a gene
gen_parameters.mutation_p_delete: float = 0.6                     # Probability of mutation deleting a gene
gen_parameters.allow_identical: bool = True                       # Offspring may be identical to any parent in prev
                                                                    generation

#Baseline:
gen_parameters.keep_baseline: bool = True                         # Baseline, if any, is always kept in population for
breeding
gen_parameters.boost_baseline: bool = False                       # Baseline is boosted to have higher probability of
breeding
gen_parameters.boost_baseline_only_co: bool = True                # Baseline is boosted for crossover selection, not
mutation



#Fitness and Hash:
gen_parameters.fitness_threshold: float = 100000.0                # Finish when best fitness is over this threshold
gen_parameters.hash_table_size: int = 100000                      # Size of hash table
gen_parameters.rerun_fitness: int = 0                             # 0-run only once, 1-according to prob, 2-always
                                                                  # Gets fitness from hash table if possible, otherwise
from
                                                                    simulation
                                                                  #rerun = 0 means never rerun
                                                                  #rerun = 1 means rerun with diminishing probability
                                                                  #rerun = 2 means rerun always
                                                                  #rerun = n with n>=3 means 3 reruns always
```

Appendix E Final Parameters of the Genetic Programming Algorithm for the Crayler Simulation for Two Pallets

```
########################################
# Parameters for Genetic Programming
########################################

#Population and Generations:
gen_parameters.ind_start_length: int = 8                              # Start length of initial genomes
gen_parameters.min_length: int = 2                                    # Minimum length of individual
gen_parameters.n_population: int = 8                                  # Number of individuals in population
gen_parameters.n_generations: int = 400                               # Maximum number of generations

#Mutation, Crossover and Selection:
gen_parameters.f_crossover: float = 0.5                               # Fraction of parent pool selected for crossover
gen_parameters.n_offspring_crossover: int = 2                         # Number of offspring from crossover per parent
gen_parameters.replace_crossover: bool = True                         # Crossover replaces subtree at receiving genome or
inserts
gen_parameters.f_mutation: float = 0.7                                # Fraction of parent pool selected for mutation
gen_parameters.n_offspring_mutation: int = 2                          # Number of offspring from mutation per parent
gen_parameters.parent_selection: int = GeneticSelectionMethods.RANK   # Selection method for parents
gen_parameters.survivor_selection: int = GeneticSelectionMethods.RANK # Selection method for survival
gen_parameters.f_elites: float = 0.1                                  # Fraction of population that survive as elites
gen_parameters.f_parents: float = gen_parameters.f_elites             # Fraction of parents that may survive to next generation
gen_parameters.mutate_co_offspring: bool = False                      # Offspring from crossover may also be mutated
gen_parameters.mutate_co_parents: bool = True                         # Parents for crossover may also be mutated
gen_parameters.mutation_p_add: float = 0.2                            # Probability of mutation adding a gene
gen_parameters.mutation_p_delete: float = 0.6                         # Probability of mutation deleting a gene
gen_parameters.allow_identical: bool = True                           # Offspring may be identical to any parent in prev
                                                                        generation

#Baseline:
gen_parameters.keep_baseline: bool = True                             # Baseline, if any, is always kept in population for
breeding
gen_parameters.boost_baseline: bool = False                           # Baseline is boosted to have higher probability of
breeding
gen_parameters.boost_baseline_only_co: bool = True                    # Baseline is boosted for crossover selection, not
mutation


#Fitness and Hash:
gen_parameters.fitness_threshold: float = 100000.0                    # Finish when best fitness is over this threshold
gen_parameters.hash_table_size: int = 100000                          # Size of hash table
gen_parameters.rerun_fitness: int = 0                                 # 0-run only once, 1-according to prob, 2-always
                                                                      # Gets fitness from hash table if possible, otherwise
from
                                                                        simulation
                                                                    #rerun = 0 means never rerun
                                                                    #rerun = 1 means rerun with diminishing probability
                                                                    #rerun = 2 means rerun always
                                                                    #rerun = n with n>=3 means 3 reruns always
```