Exception-Tolerant Hierarchical Knowledge Bases for Forward Model Learning

Daan Apeldoorn and Alexander Dockhorn

Abstract—This paper provides an overview of the recently proposed forward model approximation framework for learning games of the general video game artificial intelligence (GVGAI) framework. In contrast to other general game-playing algorithms, the proposed agent model does not need a full description of the game but can learn the game's rules by observing game state transitions. Based on hierarchical knowledge bases, the forward model can be learned and revised during game-play, improving the accuracy of the agent's state predictions over time. This allows the application of simulation-based search algorithms and belief revision techniques to previously unknown settings. We show that the proposed framework is able to quickly learn a model for dynamic environments in the context of the GVGAI framework.

Index Terms—Hierarchical Knowledge Bases, Forward Model Approximation, General Video Games, Belief Revision, Monte Carlo Tree Search

1 Introduction

Research on the topic of artificial intelligence in games aims to study how agents can be enabled to autonomously play games. While many developed agents focus on a single game to play, general game playing agents try to compete in multiple games. To evaluate such agents multiple general game playing frameworks were developed in which said agents can be tested without the need of adjusting an agent to a specific representation of a game's states or actions.

The General Video Game Playing Artificial Intelligence (GVGAI) framework [1] can be used to test agents in their capability of playing multiple arcade like video games using a unified description of the game's states and its rules. The associated GVGAI-competition agents offers multiple tracks each posing a unique challenge for competing agents. Both, the *Single-Player Planning-track* and the *Single-Player Learning-track* focus on the agents' game-playing capabilities. While in the Planning-track an agent needs to play the game based on the information on the current game state and the forward model, the learning-track does not provide the forward model. In contrast, in the original setting of the learning-track, agents were allowed to play three levels of a game for five minutes of training time, after which they are evaluated on two previously unseen levels. While the rules were changed and now allow submitted agents to be pre-trained, in this paper, we focus on the challenges implied by the extremely limited training time of the original rules.

Due to the restriction of not knowing the game's forward model, agents in the learning-track either relied on reinforcement learning techniques or made use of greedy heuristics for action-selection. Submitted solutions of the years 2017 and 2018 lacked competitiveness with submitted agents of the Planning-track, which, in comparison, often rely on simulation-based search methods, such as Monte Carlo Tree Search [2], Open Loop Search [3] and Rolling Horizon Evolutionary Algorithm [4], [5].

To overcome the huge performance gap between the solutions of the two tracks we proposed a novel general game learning algorithm called *forward model approximation* [6]. In contrast to reinforcement learning we do not try to learn the inherent value of an action or state-action pair, but model potential game state transitions after a chosen action was applied. By learning the model for mapping the current state and an action to a state transition the agent can predict the outcome of available actions and act accordingly, enabling it to apply search algorithms using the learned model.

In our underlying conference paper [6] we introduced the forward model approximation framework which uses Hierarchical Knowledge Bases to learn and refine a forward model. In this extended version, we validate the belief revision approach and provide extensive details on the model's learning and the agent's reasoning process as well as a more sophisticated analysis of the model parameters. The main contributions of this paper are:

- an introduction to forward model learning and the forward model approximation framework (Section 3)
- an analysis of forward model characteristics and how these can be exploited to improve forward model learning (Section 3.1)
- a validation of the used belief revision approach for Hierarchical Knowledge Bases representing learned forward models to underpin the applicability of Hierarchical Knowledge Bases in the context of forward model approximation (Section 4)
- a case study on the application of forward model learning, based on hierarchical knowledge bases in the context of the GVGAI framework (Sections 5 and 6)
- a parameter analysis to better explain the optimization and agent selection process (Section 5.2 and Section 6.2)

- *D. Apeldoorn primarily works for the Institute for Medical Biostatistics, Epidemiology and Informatics (IMBEI) in the Medical Informatics department at the University Medical Centre of the Johannes Gutenberg University Mainz, Germany, E-mail: daan.apeldoorn@uni-mainz.de*
- *A. Dockhorn is with Institute for Intelligent Cooperating Systems at the Department of Computer Science, Otto von Guericke University Magdeburg, Germany, E-mail: alexander.dockhorn@ovgu.de*

Manuscript received January 14, 2020

2 General Game Playing/Learning

Next to the many efforts put into artificial intelligence methods for various board games such as Chess, Morris, and Go, the Stanford General Game Playing competition [7] asked its participants to implement agents, which can compete in a set of previously unknown games. Developed agents needed to act based on the current state of the board and a set of simple rules. Many attempts have been made to create similar description languages for the definition of video games.

Thanks to the efforts of Tom Schaul, the Video Game Definition Language (VGDL) [8] was created on which the General Video Game AI (GVGAI) framework and its accompanying competition is based on. The GVGAI framework replicates many arcade like video games by describing the game's components, its forward model, and its visualization. Here, an agent receives a state description and a set of up to five possible actions. Those actions are named after buttons of a typical game controller and map the inputs "Up", "Down", "Left", "Right", and "Use". However, the outcome of each of the agent's actions is completely dependent on the rules of the game, such that a suitable policy needs to be learned for each problem individually.

In recent years the competition offered multiple tracks, which focus on different aspects of general video game playing. The next sections quickly summarize the single player game playing and the learning track as well as submissions therein. In the remainder of this paper we will focus on recent developments in context of the learning track.

2.1 GVGAI - Single Player Game Playing Track

The currently most popular track is the single player game playing track. Here, a forward model is provided to the agents, which can be used to simulate the outcome of an action and returns the resulting state and reward.

In 2018 a total of 14 submissions entered the competition (including 4 sample submissions provided by the competition hosts). The currently best performing agent, Yolobot [9], uses a mix of BFS and MCTS. While the former is used in games, which include deterministic state transitions, the latter is applied to all non-deterministic games. Additionally, the algorithm identifies reachable objects and rates its interest on them. Either BFS or MCTS is used to find a suitable path to potential targets, while avoiding any dangers.

Due to the inclusion of non-deterministic games, the application of MCTS variants such as Open Loop MCTS (OLMCTS) were studied in multiple works (see a summary of agents in [3]). OLMCTS and other tree searching algorithms such as Open Loop Expectimax Tree Search are able to quickly sample possible action sequences and evaluate their outcome [9].

Next to tree search algorithms, the search of action sequences based on genetic algorithms showed to be a popular choice. Methods such as the Rolling Horizon Algorithm [10] evolve short action sequences and evaluate their outcome based on a scoring function similar to OLMCTS.

The overall performance of agents in the game playing track is already very good. Despite being confronted with a previously unknown game, the agents are often able to win a game or at least find action sequences, which yield a high score. Due to the tracks rules not much work has been done on the analysis of reinforcement learning agents. While these may be able to choose actions much quicker than search-based agents, they also need to be extensively pretrained. Nevertheless, experiments on Atari 2600 games have shown that deep reinforcement learning-based agents are able to play a wide range of arcade games with impressive performance [11].

2.2 GVGAI - Single Player Game Learning Track

In contrast to the game playing track, the difficulty of the game learning track is much higher due to the absence of a forward model. All previously discussed methods of the game-playing track heavily rely on such a forward model, which enables the agent to run simulations for hypothetical action sequences. Since the training time is also limited to 5 minutes of playtime, reinforcement learning algorithms cannot yet effectively be applied. Therefore, new algorithms need to be found for solving problems of this domain.

In the previous years multiple agents have been submitted with varying success, however, not much information is available on them. The submission by Ercüment İlhan [12] uses an MCTS agent, which was enhanced with an online on-policy temporal-difference learning method, called *true online Sarsa(λ)* [13]. Here, the agent optimizes its estimation of the state-action value function by continuously revising it based on the repeated interactions with the game environment. This agent performed best in the training set, but placed 5th out of 6 in the test game set (slightly ahead of an adaptation of the Yolobot from the game playing track). While in theory this learning approach should be able to detect useful action sequences, the available learning time is too short to effectively do so. Nevertheless, this approach can become in handy, in case enough training samples can be gathered during the training time or observations can be generalized across different game-states.

Surprisingly the random controller provided by the competition organizers performed second best in the evaluation set of 2018 and became first place in 2019. This shows that well-known methods for value estimation or policy improvement do not work at their full potential using the limited information in this track. Short time frames for learning the game's rules ($\leq$ 5 minutes of playing time in the former original competition specification) and decision-making (40ms per time step) drastically limit the applicability of iterative methods or long search procedures. Hence, there is still more room for improvement.

Our work contributes to overcome the huge performance gap between agents of the game playing and game learning track and to the idea that machine learning combined with methods from symbolic knowledge representation can be combined to create an appropriate agent model. This work will show how the introduction of *approximated forward models* and efficient learning and operation schemes can help us in developing new kinds of agents. Learning these forward models cannot just be of value for simulation-based search approaches, but also benefit reinforcement learning agents by enabling model-based reinforcement learning. These would be able to refine their policy based on simulated interactions with the approximated forward model after the training time on the real environment is over.

3 FORWARD MODEL LEARNING

Agents for general game-playing often rely on the availability of a forward model. Since played games are unknown at the time of developing the agent, no specialized rules can be implemented. As a result, agents are required to learn how to play a given game by observation.

Reinforcement learning agents tackle this challenge by trying to estimate the value of available actions. While this approach has been successfully applied to games of the Atari Learning Environment [14], [15], the experiments have also shown that the agent requires many trials until reasonable game-playing performance can be achieved. The trial-and-error behavior can be avoided when providing the agent with an internal model of the world. Reasoning about future game-states, a process which is also referred to as "imagining", allows the agent to spot and avoid adverse consequences by predicting the game-state several steps ahead [16]. In an analogous manner, an internal model of the world enables agents to apply search algorithms and evaluate the value of its actions using simulations of its environment.

As the term suggests, forward model learning focuses on learning a reliable model of the agent's environment by interacting with it and observing the consequences. Using a data set of these observations the learning task can be represented as a classification or regression problem in which the current state and action are mapped to the next state.

Proposed approaches generally differ in the way they observe and process the environment's state as well as how they represent the learned forward model. Pixel-based state observations have been used in conjunction with latent imagination to learn complex motion behaviors [17] as well as controllers for a top-down racing game and Doom [18]. Using an object-based representation, several works have presented forward models for games of the GVGAI framework [6], [19] and Mario [20]. In such an object-based observation the entirety of observable sensor values is grouped into one vector per object in the scene. In case such a grouping is missing, a study by Dockhorn and Kruse [21] suggests that it can be inferred through dependency analysis. Recent studies on spatially structured representations (such as a tile-map) include the games Sokoban [22] and the Game of Life [23]. These approaches achieve a higher sampling efficiency by applying a convolutional approach in which each tile is predicted given its local neighborhood. Even without such a structured representation, forward models have been learned for several motion control tasks [24], [25] and specifically robot control [26].

Interesting characteristic of forward model-based methods include but are not limited to:

- **generalizeablity:** models can be applied across states and may even apply to previously unobserved states.
- **transfer learning:** the model can be independent of the underlying task enabling transfer learning across multiple games, e.g. general models for jump-and-run games or top-down view grid-based games.
- **active search:** after wrong predictions the agent may be able to spot its own weaknesses in the prediction of future events. This may enable self-motivated or active learning methods in which the agent actively seeks new experiences in the environment to refine its model.
- **risk avoidance:** Due to the forward model the agent is able to search for possible positive outcomes, while avoiding adverse consequence of trial-and-error.

3.1 Fundamentals of Forward Model Learning

In general the response of the environment can be modelled as a probability distribution over all previous interactions with the agent [27].

$$P(r_{t+1}, s_{t+1}|s_0, a_0, \ldots, s_t, a_t)$$

Learning or storing such a specific probability distribution is often infeasible due to the exponential growth over time and the large number of possible states, actions, and rewards.

Therefore, reinforcement learning often focuses on Markov Decision Processes, in which we assume that the successor state and reward are only dependent on the latest interaction between the agent and the environment [28]. In other words the probability distribution for the successor state reduces to:

$$P(s_{t+1}|s_0, a_0, \ldots, s_t, a_t) = P(s_{t+1}|s_t, a_t)$$

and the reward function maps the last state action pair to a provided reward:

$$R(s_t, a_t) = r_{t+1}$$

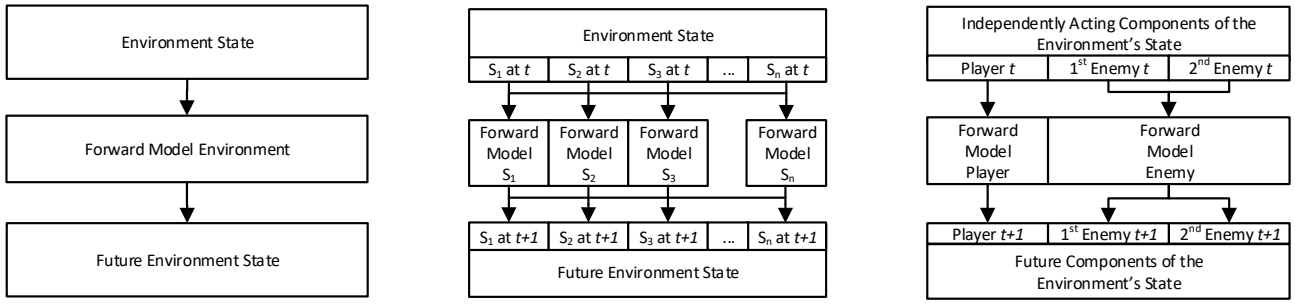
3.2 Basic Model

In the context of *exception-tolerant Hierarchical Knowledge Bases* (HKBs), an agent is usually considered to be equipped with n sensors through which it can perceive its current state in the environment (e. g., a game). Furthermore, the agent is able to perform actions from a predefined action space (e. g., the keys to be pressed on the controller). The agent is also able to perceive how good the performed actions were, in form of (numeric) rewards. The perceived rewards can then be used to learn a weighted state-action pair representation where the highest weight determines which action has to be performed, given a perceived state.

More formally, as described in [29], in such a representation, a state s is an element of a multi-dimensional state space $\mathbb{S} = \mathbb{S}_1 \times ... \times \mathbb{S}_n$ where n is the number of the agent's sensors and every $\mathbb{S}_i$ is a set of possible values of the corresponding sensor. Furthermore, the agent selects actions from a predefined action set $\mathbb{A}$ and the learned weights are stored in a multi-dimensional matrix $\hat{Q} = (q_{s_1,\ldots,s_n,a})$ with $s_i \in \mathbb{S}_i$ and $a \in \mathbb{A}$. The weights can be learned by different machine learning approaches, provided that the learning approach results in a representation such that, given a state, the highest weight determines the best action to be selected (i. e., $a^{\max}_{s_1,\ldots,s_n} = \arg\max_{a' \in \mathbb{A}} q_{s_1,\ldots,s_n,a'}$).

After selecting an action the environment responds with a reward and a successor state s_{t+1}. Our target will be to construct a classifier, hereinafter referred to as *approximated forward model*, which approximates the distribution $P(s_{t+1}, r_{t+1}|s_t, a_t)$ based on a history of previous interactions with the environment.

Such an *approximated forward model* can be split into multiple sub-components in case the multi-dimensional representation $\mathbb{S}$ consists of independent components. If this



(a) End-To-End Forward Model (b) Decomposed Forward Model (c) Object-based Forward Model

Fig. 1. Comparison of forward model architectures: (left) single end-to-end model for the whole environment; (middle) forward model-ensemble consisting of a sub-model for each sensor of the environment; (right) forward model-ensemble consisting of a sub-model for each type of entities [21]

is the case, a complete reconstruction of the state space $\mathbb{S}$ can be achieved by modelling each independent component separately. See Figure 1 for a comparison of popular forward model types.

The following subsections introduce HKBs, which will be used for learning and later also for revising such classifiers (using belief revision techniques) during continuous interactions with the games environment.

4 Hierarchical Knowledge Bases

This section briefly outlines the preliminaries and components needed for the agent model learning an *approximated forward model* based on *exception-tolerant Hierarchical Knowledge Bases* (HKBs), which will be introduced in Section 5. At first, the basics needed for this approach will be explained, for which it is useful to define HKBs and how they can be learned (Subsections 4.1 and 4.2). These sections will closely follow several preliminary works, especially [6], [29], [30] and [31]. After that, the reasoning algorithm defined on HKBs will be briefly summarized, following [31] (Subsection 4.3). Subsequently, some *modifications* on HKBs and the learning process (Subsection 4.4) to fit the needs of our agent model will be explained. Finally, a belief revision approach for HKBs will be described and validated (Section 4.5).[1]

4.1 Foundations of HKBs

This section provides an overview and the main definitions needed to understand the basic idea of HKBs as well as the basic algorithm how they can be learned from weighted state-action pairs by closely following [29], [30], [31] and [6]. For more details on HKBs, the reader should refer to the original literature mentioned here.

4.1.1 Overview

An HKB is a knowledge representation approach which allows for efficient rule-based knowledge representation in a human readable and generalizing way. It can be used, e. g., for describing agent behavior.

1. Implementations of relevant algorithms for integrating learning, reasoning and belief revision based on HKBs can be found in the open-source toolbox *InteKRator* [32].

The main idea is to model knowledge with as few as possible rules, by making use of rough general rules (involving only a few sensory information) with exceptions. Exceptions are represented by more specific rules (involving more sensory information) and by this, also exceptions from exceptions (i. e., 2^{nd} and higher order exceptions) are possible. This allows for representing knowledge very compactly by only relying on exception rules where needed.

4.1.2 Technical Details

An HKB consists of rules which are organized on different levels of abstraction. An HKB can handle multiple rules per level and the rules also comprise weights. According to [31], two different kinds of states and two different kinds of rules need to be distinguished:

Definition 1 (Complete States/Partial States) *A complete state is a conjunction $s := s_1 \wedge ... \wedge s_n$ of all values $s_i \in \mathbb{S}_i$ currently perceived by the agent's sensors, where n is the number of sensors and every perceived sensor value $s_i \in \mathbb{S}_i$ of the corresponding sensor value set $\mathbb{S}_i$ is assumed to be a fact in the agent's current state. A partial state is a conjunction $s := \bigwedge_{s' \in S} s'$ of a subset $S \subset \{s_1, ..., s_n\}$ of the sensor values of a complete state.*

Definition 2 (Complete Rules/Generalized Rules) *Complete rules and generalized rules are of the form $p_\rho \Rightarrow a_\rho \ [w_\rho]$, where p_ρ is either a complete state (in case of a complete rule) or a partial state (in case of a generalized rule). The conclusion $a_\rho \in \mathbb{A}$ is an action of the agent's action space $\mathbb{A}$ and $w_\rho \in [0, 1]$ is the rule's weight.*

Thus, according to these two definitions, *complete rules* map *complete states* to actions and *generalized rules* map *partial states* to actions.

An HKB can now be defined as follows:

Definition 3 (Exception-Tolerant Hierarchical Knowledge Base) *An exception-tolerant Hierarchical Knowledge Base (HKB) is an ordered set $\mathcal{KB} := \{R_1, ..., R_{n+1}\}$ of $n + 1$ rule sets, where n is the number of sensors (i. e., the number of state space dimensions). Every set $R_{j<n+1}$ contains generalized rules and the set R_{n+1} contains complete rules, such that every premise $p_\rho = \bigwedge_{s \in S_\rho} s$ of a rule $\rho \in R_j$ (with $S_\rho \subseteq \{s_1, ..., s_n\}$ and $s_i \in \mathbb{S}_i$) is of length $|S_\rho| = j - 1$.*

According to Definition 3, the set R_1 contains the most general rules (with empty premises) and the set R_{n+1} contains the most specific (i. e., complete) rules.

For the relations of rules, the term of *(needed) exception* will be used, according to the following definition (cf. [29]):

Definition 4 (Needed Exception) *A rule $\rho \in R_{j>1}$ is an exception to a rule $\tau \in R_{j-1}$ with premise $p_\tau = \bigwedge_{s \in S_\tau} s$, action a_τ as conclusion and weight w_τ, if $S_\tau \subset S_\rho$ and $a_\rho \neq a_\tau$. The exception is needed, if no other rule $\upsilon \in R_{j-1}$ exists with premise $p_\upsilon = \bigwedge_{s \in S_\upsilon} s$ and action a_υ as conclusion where $S_\upsilon \subset S_\rho$ and $a_\upsilon = a_\rho$ and $w_\upsilon > w_\tau$.*

4.2 Learning HKBs

An HKB can be extracted from a weighted state-action pair representation $\hat{Q}$ (that is learned, e. g., through a reinforcement learning technique or by simply counting relative frequencies) using the following approach which was originally introduced in [30], closely following [31] here.

The input is a weighted state-action pair representation $\hat{Q}$ and the output is an HKB $\mathcal{KB}^{\hat{Q}}$ which reflects the knowledge contained in $\hat{Q}$ by performing the following steps:[2]

1) *Initial creation of rule sets:*
   In the first step, the multiple abstraction levels $R_1, ..., R_{n+1}$ of the knowledge base are initially filled with rules. The weights of generalized rules are created by averaging the weights in $\hat{Q}$ over the missing dimensions.
2) *Removal of worse rules:*
   In all sets R_j, a rule $\rho \in R_j$ is removed, if there exists another rule $\sigma \in R_j$ with the same partial state as premise having a higher weight.
3) *Removal of worse more specific rules:*
   In all sets $R_{j>1}$, a rule $\rho \in R_j$ with premise $p_\rho = \bigwedge_{s \in S_\rho} s$, conclusion a_ρ and weight w_ρ is removed, if there exists a more general rule $\sigma \in R_{j'<j}$ with premise $p_\sigma = \bigwedge_{s \in S_\sigma} s$ where $S_\sigma \subset S_\rho = \{s_1, ..., s_{j-1}\}$ and weight $w_\sigma \geq w_\rho$.
4) *Removal of too specific rules:*
   In all sets R_j, a rule $\rho \in R_{j>1}$ with premise $p_\rho = \bigwedge_{s \in S_\rho}$ and conclusion a_ρ is removed, if there exists a more general rule $\sigma \in R_{j'<j}$ with the same action $a_\sigma = a_\rho$ as conclusion and with premise $p_\sigma = \bigwedge_{s \in S_\sigma} s$ where $S_\sigma \subset S_\rho = \{s_1, ..., s_{j-1}\}$ and if ρ is not a *needed exception* to a rule $\tau \in R_{j-1}$.
5) *Optional filter step:*
   Optionally, filters may be applied to filter out further rules which are, e. g., helpful to explain the knowledge contained in $\hat{Q}$ through the optimal found policy so far, but which are not needed for reasoning later.

After performing these steps on $\hat{Q}$, the knowledge base $\mathcal{KB}^{\hat{Q}}$ comprises all sets $R_j \neq \emptyset$ with the extracted rules representing the implicit knowledge contained in the learned weights of $\hat{Q}$ in a compact way.

2. For performance reasons, only state-action pairs can be considered here that contribute to the best policy found by the preceding learning process; an implementation of a faster algorithm for learning HKBs from state-action sequences can be found in [32].

4.3 Reasoning with HKBs

This section briefly summarizes the basic idea of the efficient reasoning algorithm on HKBs which was first introduced in [30]. The summary closely follows [31]:

Given an HKB $\mathcal{KB}$ together with the perceived state $s = s_1 \wedge ... \wedge s_n$ (with the perceived sensor values $s_1, ..., s_n$), the reasoning algorithm $\mathfrak{R}(\mathcal{KB}, s)$ searches $\mathcal{KB}$ upwards (starting from the bottom-most level $R_{n+1} \in \mathcal{KB}$) for the first rule whose premise is satisfied. This rule is then applied and the concluding action $a \in \mathbb{A}$ is returned (see [30] for details). By this, the algorithm selects the most specific rule that fits to the perceived sensor values and falls back to the next more unspecific rule (which can be considered a heuristic), in case no more specific rule with a fitting premise could be found.

4.4 Modifications for Our Agent Model

In this section, the original HKB approach (as described in Section 4) will be modified to fit the needs of our agent model, closely following [6].

A knowledge representation based on exceptions which are layered on several levels of abstraction is a rather useful approach to gain a compact representation of the knowledge about an environment like a game. This knowledge can also be exploited during a learning process which has been demonstrated in [30], [31]. Nevertheless, according to the original GVGAI competition specification, our learning agent is supposed to work in *multiple* levels of a game and the agent furthermore only sees three out of five levels in the training phase. Thus, the agent should be able to learn the *general mechanics* of the game rather than optimizing its behavior for a single level.

For this purpose, we modify the definitions of HKBs (especially Definition 2) such that rules no longer represent a mapping of a state to an action but a mapping of a state and an action which was performed in that state to a resulting subsequent state. More formally, rules contained in the HKB are now defined as follows:

Definition 5 (Modified Complete/Generalized Rules) *The modified complete rules and generalized rules are of the form $p_\rho \wedge a \Rightarrow p'_\rho\ [w_\rho]$, where p_ρ is either a complete state (in case of a complete rule) or a partial state (in case of a generalized rule), $a_\rho \in \mathbb{A}$ is an action of the agent's action space $\mathbb{A}$, p'_ρ represents (a part of) the changes in the subsequent state (resulting from action a performed in state p_ρ) and $w_\rho \in [0, 1]$ is the rule's weight. The values used for any state change representation p'_ρ can be considered disjoint from the values used for any rule's premise.*

Note that since the creation of HKBs from data is computationally rather expensive (cf. footnote 2), in case of our agent model, we will only consider small subsets $\mathbb{S}' \subset \{\mathbb{S}_1, ..., \mathbb{S}_n\}$ of the agent's state space dimensions for p_ρ and p'_ρ in Definition 5. This results in several smaller HKBs where every HKB represents a certain aspect of the agent's collected knowledge about the environment. Furthermore, a merging technique will be used to gain an HKB for higher dimensional state spaces by merging multiple smaller HKBs. (For details see Section 5.)

In addition, the extraction algorithm described in Section 4.2 will be extended by the following filter at the

end of Step 5: All rules $\rho \in R_{j>1}$ whose premises do not contain an action will be removed.

In the following example (extended from [6]), the idea of such modified HKBs will be explained in the context of the game *Butterflies* from the GVGAI competition framework [1].

Example 1 (Butterflies) *We consider the game Butterflies from the GVGAI competition framework [1], where an agent has to collect butterflies by touching them (see upper left part of Figure 2): If a butterfly is collected, the agent's current score is increased by 2. To learn knowledge about scoring, movement and winning in the context of the game, the agent's surrounding objects, the game's end state (win/lose), etc. are considered as subsets of the state space dimensions. The agent's action space is defined as* $\mathbb{A} := \{\text{Up}, \text{Down}, \text{Left}, \text{Right}, \text{Use}\,(, \text{None})\}$. *The HKBs, which were learned after a short training phase and which comprise the knowledge about scoring, movement and winning (i. e., in which states which actions lead to which changes regarding the respective aspect of the game), are also shown in Figure 2.*

$\mathcal{KB}_{\text{move}}$:

x−1, y±0 [0.25]	R_1
Down → x±0, y+1 [0.75] Up → x±0, y−1 [0.75] Right → x+1, y±0 [0.75] Left → x−1, y±0 [0.75]	R_2
above0 ∧ Up → x±0, y±0 [1.0] below3 ∧ Right → x±0, y±0 [1.0] below0 ∧ Down → x±0, y±0 [1.0] left0 ∧ Left → x±0, y±0 [1.0] left3 ∧ Right → x±0, y±0 [1.0] right0 ∧ Right → x±0, y±0 [1.0]	R_3

$\mathcal{KB}_{\text{score}}$:

score±0 [0.9375]	R_1
above5 ∧ Up → score+2 [1.0] below5 ∧ Down → score+2 [1.0] left5 ∧ Left → score+2 [1.0] right5 ∧ Right → score+2 [1.0]	R_3

$\mathcal{KB}_{\text{win}}$:

gamestate0 [1.0]	R_1

Fig. 2. Excerpt from the First Level of the GVGAI Game *Butterflies* with Corresponding HKBs of the Agent's Knowledge About the Different Aspects *Scoring*, *Movement* and *Winning* After a Short Training Phase (extended from [6])

The HKBs resulting from Example 1 (see Figure 2) can be read as follows: In case of $\mathcal{KB}_{\text{score}}$, according to the single rule $\text{score}\pm 0$ on the most general level R_1 (which has an empty premise), the agent learned that *in general* (when no action is performed), no score changes are expected. This covers most of the cases as indicated by the high weight. According to the four rules on level R_3 of $\mathcal{KB}_{\text{score}}$, the agent learned that if an object with id 5 (i. e., a butterfly) is perceived above, below, to the left or to the right of the agent and the agent performs an action in the corresponding direction of the object, then the score is increased by 2. The weights of 1.0 indicate that this should happen in all cases when perceiving these objects and performing these actions. Level R_2 does not contain any rules, thus there are no relevant exceptions from the most general rule on level R_1 in this game that are only based on actions (without considering any surrounding objects). Furthermore, level R_4 is also empty, since there are no relevant exceptions from the rules on level R_3 that additionally involve the orientation of the agent. The other two HKBs $\mathcal{KB}_{\text{move}}$ and $\mathcal{KB}_{\text{win}}$ can be read in the same way. $\mathcal{KB}_{\text{win}}$ only contains one single rule stating that the game ends always in *gamestate0* (which means *lose*). This is the case, since the game was never won by the agent during random exploration in the short training phase. In case some learned rules turn out to be wrong later, they can be revised by a revision approach without the need of relearning the whole model (see Section 4.5). (See [32] for implementations of learning, reasoning and belief revision algorithms based on HKBs.)

Input: HKB $\mathcal{KB} = \{R_1, ..., R_{n+1}\}$, state with action $s_a = s_1 \wedge ... \wedge s_n \wedge a$, successor state information s'

Output: Revised HKB $\mathcal{KB}' = \{R'_1, ..., R'_{n+1}\}$

```
% If a wrong conclusion is inferred for the given
% state-action conjunction and successor state info
if s' ≠ ℜ(KB, s_a) then

    % Add new exception, if causative rule not on
    % most specific level...
    if ρ_{s_a} ∉ R_{n+1} then % ρ_{s_a} is the rule firing for
                              % providing ℜ(KB, s_a)
        R_{n+1} := R_{n+1} ∪ {s_a → s' [1.0]}

    % ...else remove/exchange existing exception
    else
        R_{n+1} := R_{n+1} \ {ρ_{s_a}}
        if s' ≠ ℜ(KB, s_a) then
            R_{n+1} := R_{n+1} ∪ {s_a → s' [1.0]}
        end if
    end if
end if
```

Algorithm 1. Belief Revision for HKBs: If a wrong conclusion inferred by $\mathfrak{R}$ is caused by a rule not on level R_{n+1}, a new exception rule is added on R_{n+1}; otherwise the rule providing the wrong conclusion is removed or exchanged.

4.5 Belief Revision for HKBs

Unlike *relearning* knowledge that is once gained by subsymbolic machine learning approaches (e. g., during the training phase of a game), a much more efficient solution could be a symbolic revision approach used on a previously learned knowledge base. By this, the learned knowledge can be expanded or changed immediately on a symbolic level instead of relearning statistically on a subsymbolic level. For this purpose, this section first provides an algorithm for the belief revision approach for HKBs used in [6] (Section 4.5.1). The described approach is a simple attempt to realize revision for HKBs, which allows the agent to revise the learned knowledge represented by an HKB efficiently, in case the environment changes. In addition, the approach will be evaluated here against the six basic AGM postulates (e. g., [33]) known from belief revision to underpin the soundness of the approach (Section 4.5.2).

4.5.1 A Simple Belief Revision Approach

We consider a learned HKB $\mathcal{KB}$ with the modifications described in Section 4.4 and the reasoning algorithm $\mathfrak{R}$ described in Section 4.3: Given a state representation s and an action a, and given that the representation of a subsequent state s' inferred through the HKB by $\mathfrak{R}$ is *inconsistent* with the corresponding representation of the actual perceived subsequent state s'_{per} of the agent (i. e., $s' \neq s'_{\text{per}}$), the basic processing of the revision algorithm is provided Algorithm 1:

By this, the learned knowledge about the game mechanics can be quickly adapted to changes in the environment (e. g., scoring distributions, object localizations or even new kinds of objects) in case the agent is being confronted with new levels.

4.5.2 Verifying Against the Basic AGM Postulates

When a novel belief revision approach is considered, a common question is, whether the proposed approach is reasonable in the sense that it leads to an appropriate state of the represented knowledge after the revision is performed. The AGM postulates, named after Alchourrón, Gärdenfors, and Makinson (e. g, [33]) define some general criteria for this purpose. In this section, the proposed revision approach for HKBs will be evaluated against the six basic AGM postulates for revision, according to the more modern representation as provided, e. g., in [34] (using $\mathcal{KB}$ for the knowledge to be revised, $*$ as revision operator, η for the new information and $+$ is an expansion operator simply adding the new information without any efforts in preserving consistency):

1) *Closure:* $\mathcal{KB} * \eta$ is a deductively closed set of beliefs.
2) *Success:* $\eta \in \mathcal{KB} * \eta$
   (after the revision operation, the new information must be contained in the revised knowledge)
3) *Inclusion:* $\mathcal{KB} * \eta \subseteq \mathcal{KB} + \eta$
   (the revised knowledge must not comprise more information as if the new information simply had been added)
4) *Vacuity:* If $\neg\eta \notin \mathcal{KB}$ then $\mathcal{KB} * \eta = \mathcal{KB} + \eta$
   (if the new information does not contradict what is already known, it can simply be added)
5) *Consistency:* If η is consistent, then also $\mathcal{KB} * \eta$ must be consistent.
6) *Extensionality:* If η_1 and η_2 are equivalent pieces of information, then $\mathcal{KB} * \eta_1 = \mathcal{KB} * \eta_2$.

In the following, it will be shown that the proposed revision approach for HKBs can be considered valid against the basic AGM postulates.

Since the original AGM postulates are defined to be applied on a deductively closed set of general logical formulas being revised (or contracted) with another logical formula representing the new information (or the information to be forgotten, respectively), some preliminary adaptions to the basic AGM postulates for revision are necessary in some cases, to be able to apply them to revision on HKBs.

As first preliminary considerations, it can be remarked that the revision algorithm (Algorithm 1) consists of three operations to change the HKB $\mathcal{KB}$:

- *adding a new exception* (which adds a new exception rule on the most specific level $R_{n+1} \in \mathcal{KB}$),
- *removing an existing exception* (which removes a wrong exception rule on the most specific level $R_{n+1} \in \mathcal{KB}$)
- *exchanging an existing exception* (which replaces the conclusion of a wrong exception rule on the most specific level $R_{n+1} \in \mathcal{KB}$ with the expected correct conclusion).

After *removing an existing exception*, it is first checked whether the correct conclusion can be inferred through the reasoning algorithm $\mathfrak{R}$. Only if this is not the case, *adding a new exception* is performed after the removal. Thus, *exchanging an existing exception* only leads to the same resulting knowledge base than an execution of *removing an existing exception* followed by *adding a new exception* if $\mathfrak{R}$ did not provide the correct conclusion after the removal. For this reason, *exchanging an existing exception* will be handled as a separate case in the following.

Let now $\mathcal{KB}$ be an HKB containing rules according to Definition 5 and let $\mathcal{KB}'$ be the HKB revised by Algorithm 1, then the belief revision algorithm can be considered valid against the basic AGM postulates for the following reasons:

1) *Closure:* Since the values of the rules' premises and conclusions can be considered *disjoint* (according to Definition 5), conclusions cannot contribute to further new inferences (except for the topmost rule(s) on level R_1). Thus, every HKB can always be considered *closed* in this sense.

2) *Success*: For the second postulate, it will be argued that after the revision approach, the reasoning algorithm $\mathfrak{R}$ (as described in Section 4.3) will return the new correct conclusion c for a given state-action pair $s_a = s_1 \wedge ... \wedge s_n \wedge a$, where c was introduced either by *adding a new exception* or by *exchanging an existing* one, or c stems from a rule on a level $R_{j<n+1}$ after *removing an existing exception* on level R_{n+1} which provided the wrong conclusion:
   - In case of *adding a new exception* for s_a on level R_{n+1}, the corresponding added rule ρ is of the form $s_1 \wedge ... \wedge s_n \wedge a \rightarrow c$ where c is the new correct conclusion. When applying $\mathfrak{R}(\mathcal{KB}', s_a)$, $\mathfrak{R}$ will search the revised HKB $\mathcal{KB}'$ starting on the revised level R'_{n+1} for the first rule whose premise is fulfilled by s_a (as described in Section 4.3). Since a new rule is only added if there is no other rule $\sigma \in R_{n+1}$ with premise s_a (otherwise the belief revision would have been done by *exchanging an existing exception*) and since the premise of ρ represents a complete state, the new rule ρ will be the only rule on the revised bottom most level R'_{n+1} which is found by $\mathfrak{R}$ and its conclusion c will be correctly returned.
   - In case of *removing an existing exception* for s_a on level R_{n+1}, a new rule with the correct conclusion c is only added if $\mathfrak{R}(\mathcal{KB}', s_a)$ does not infer the correct conclusion c from a rule on a level $R_{j<n+1}$. Thus, either $\mathfrak{R}(\mathcal{KB}', s_a)$ returns c successfully after the removal or this case falls back to the case of *exchanging an existing exception*.
   - In case of *exchanging an existing exception* for s_a on level R_{n+1}, the conclusion of the corresponding firing rule found by $\mathfrak{R}$ on level R_{n+1} will be exchanged by the correct conclusion c. Since only the conclusion of that rule will be exchanged, $\mathfrak{R}(\mathcal{KB}', s_a)$ will still find the same rule after the exchange and return the exchanged conclusion c (which is the correct one).

3) *Inclusion*: For the inclusion postulate, it will be shown that in case of *removing an existing exception* and in case of *exchanging an existing exception* no more conclusions than in case of *adding a new exception* on level R_{n+1} can be inferred after revision.
   The case of *adding a new exception* implies that no other rule with the same premise existed on level R_{n+1} before (otherwise it would have been be the case of *exchanging an existing exception*). Thus, since the new exception rule is added on the most specific level R_{n+1} with the premise according to the perceived complete state $s_a = s_1 \wedge ... \wedge s_n \wedge a$ and the new correct conclusion c, it is

ensured that in case of perceiving s_a, the new conclusion c will be inferred by $\mathfrak{R}(\mathcal{KB}', s_a)$ and no other inferences will be affected (since s_a is a complete state).

- In case of *removing an existing exception* for the complete state s_a, no other rule is added if $\mathfrak{R}(\mathcal{KB}', s_a)$ returns the correct conclusion c after the removal. Since the exception is removed on the most specific level R_{n+1} (according to Section 4.5.1), this changes the inference only for exactly the complete state s_a. All other perceived states will result in the same conclusion than before the revision. Therefore, no further new conclusions can be inferred as in case the new exception rule simply would have been added on level R_{n+1} for s_a.
- In case of *exchanging an existing exception* for the complete state s_a: Since the exception rule with the corrected conclusion c is located on the revised level R'_{n+1} (according to Section 4.5.1), only the inference for exactly the complete state s_a is affected. Thus, no further new conclusions can be inferred by $\mathfrak{R}$ from $\mathcal{KB}'$.

4) *Vacuity*: For the vacuity postulate, it will be shown that in case no *conflicting* exception rule exists (i. e., there is no rule on level R_{n+1} with a premise corresponding to the perceived state $s_a = s_1 \wedge ... \wedge s_n \wedge a$ that provides a wrong conclusion), both cases *removing an existing exception* and *exchanging an existing exception* lead to the same result than simply *adding a new exception*. Only the level R_{n+1} needs to be considered here, since if a wrong conclusion stems from a rule on a level $R_{j<n+1}$, then a corresponding new exception rule with the correct conclusion for the complete state s_a will be added on level R_{n+1} (which obviously did not exist before—otherwise it would have been the one providing the wrong conclusion).
   More concretely, the vacuity postulate must hold, if for the state $s_a = s_1 \wedge ... \wedge s_n \wedge a$ to be revised, no rule exists on level R_{n+1} with s_a as premise and conclusion c' unequal to the correct conclusion c.
   According to Section 4.5.1, the precondition of the belief revision algorithm for *removing an existing exception* or *exchanging an existing exception* is, that the wrong conclusion stems from a rule on level R_{n+1}. Thus, if the wrong conclusion stems from level $R_{j<n+1}$, the revision algorithm simply falls back to *adding a new exception* in both cases.

5) *Consistency*: By definition, an HKB cannot be inconsistent, since even if multiple conclusions can be inferred for a given state $s_a = s_1 \wedge ... \wedge s_n \wedge a$, this semantically expresses that the conclusions are *equivalent* and not *contradictory* for the given state. Thus, every HKB will be consistent in that sense.

6) *Extensionality*: According to Section 4.5.1, revision is always done for a perceived complete state $s_a = s_1 \wedge ... \wedge s_n \wedge a$ for which a wrong conclusion is returned by $\mathfrak{R}(\mathcal{KB}, s_a)$. Thus, it has to be shown that for any equivalent information $s'_a \equiv s_a$ (i. e., any permutation of the operands $s_1, ..., s_n, a$), the revision will result in the same HKB $\mathcal{KB}' = \mathcal{KB}$. To verify this, it has to be shown that (i) all three operations *adding*, *removing* and *exchanging an exception* result in the same modification for s'_a and (ii) that the same modifying operation is selected for s'_a.
   For (i), this follows from the commutativity of the logical conjunction, since s'_a will be the premise of the rule to be added, or the rule to be removed or exchanged will be selected according to the rule's premise being logically equivalent (since rules are only added, removed or exchanged on the bottom most level R_{n+1} where the rules' premises are complete states).
   For (ii), this also follows from the commutativity of the logical conjunction, since (according to Section 4.5.1) all decisions whether an exception rule is *added*, *removed* or exchanged solely depend on a rule's premise being logically satisfied.
   Thus, if $\mathcal{KB}$ is being revised with an equivalent state s'_a, it follows that $\mathcal{KB}' = \mathcal{KB}$.

5 AGENT MODEL

The agent model builds on the preliminaries described in Section 3 and Section 4. First, the basic composition of the concepts required to learn the game mechanics will be summarized (Section 5.1). Consequently, this section focuses on how the action selection is realized based on these ideas and which modifications where necessary to optimize the process (5.2). An overview of the agent model is provided in Figure 3, accompanied by a supplemental higher level description summarizing the steps of the agent's process (Section 5.3).

5.1 Basics

Since the learning track of the GVGAI competition is divided into a *training phase* (on three out of five levels of each game) and an *evaluation phase* (on already known and two additional levels), the basic idea is to use the training phase to accumulate knowledge about the game mechanics in form of modified HKBs (as described in Section 4.4) and to use planning based on the gained knowledge in the evaluation phase. Furthermore, a belief revision approach is used during the evaluation phase in case new experiences are contradicting the learned knowledge of the training phase in some aspects.

5.1.1 Dividing the Learned Knowledge into Different Aspects

Games can vary widely in their game play and the goals that have to be reached to win the game. Thus, different aspects of the game mechanics are important depending on the kind of game that is played. For this purpose, the knowledge about the game mechanics will be stored in three different HKBs – one HKB for one type of knowledge representing one aspect that might be relevant for decision-making (cf. also Figure 2). The following three aspects are covered:

- $\mathcal{KB}_{\text{move}}$, *Relative Movement*: The movement depending on the relative position to other objects (e. g., obstacles like "objects of this type cannot be passed").

- $\mathcal{KB}_{\text{scr}}$, *Scoring*: Score changes depending on interactions with other objects (e. g., beneficial objects like "collecting objects of this type increases the score by X").
- $\mathcal{KB}_{\text{win}}$, *Winning/Losing*: Which kind of object interactions lead to winning or losing the game (e. g., objects that lead to winning the game when touching them).

The division of the forward model into multiple HKBs can be justified in case the game's individual components are independent from another. In this case the complete model is reconstructible from the outputs of each sub-model.

5.1.2 Fast Creation of HKBs for the Different Aspects

As mentioned in Section 4.2 and Section 4.4, creating HKBs can be computationally expensive on higher dimensional state spaces. To overcome this problem, for every HKB contained in the meta knowledge base, multiple separate HKBs are created from reduced state spaces with less dimensions. The resulting HKBs are than merged to one final HKB representing one of the three types of knowledge $\{\mathcal{KB}_{\text{move}}, \mathcal{KB}_{\text{scr}}, \mathcal{KB}_{\text{win}}\}$.

In the following, the creation of the *Scoring* HKB $\mathcal{KB}_{\text{scr}}$ will be described (the process is very similar for the other two types of knowledge in the meta knowledge base):

$\mathcal{KB}_{\text{scr}}$ reflects the knowledge about which action leads to which score change, given the orientation of the agent and the types of the objects currently surrounding it. The HKB $\mathcal{KB}_{\text{scr}}$ could be created (according to the algorithm described in Section 4.2) from the matrix $\hat{Q}_{\text{scr}} = (q_{s_{\text{above}}, s_{\text{below}}, s_{\text{left}}, s_{\text{right}}, s_{\text{ori}}, a, s_{\text{scr}}})$ with $s_{\text{above}}, s_{\text{below}}, s_{\text{left}}, s_{\text{right}} \in \mathbb{S}_{\text{obj}}$, $s_{\text{ori}} \in \mathbb{S}_{\text{ori}}$, $a \in \mathbb{A}$ and $s_{\text{scr}} \in \mathbb{S}_{\text{scr}}$, where $\mathbb{S}_{\text{obj}}$ is the set of object types, $\mathbb{S}_{\text{ori}}$ is the set of the agent's orientations, $\mathbb{A}$ is the agent's action space and $\mathbb{S}_{\text{scr}}$ is the set of score changes. Every element of $\hat{Q}_{\text{scr}}$ represents a learned relative frequency of how often an action leads to a certain score change, given the agent's orientation and the types of objects *above*, *below*, *left*, and *right* of the agent.

However, instead of creating $\mathcal{KB}_{\text{scr}}$ directly from the seven-dimensional matrix $\hat{Q}_{\text{scr}}$, as a first step, the four smaller HKBs $\mathcal{KB}_{\text{scr}}^{\text{above}}$, $\mathcal{KB}_{\text{scr}}^{\text{below}}$, $\mathcal{KB}_{\text{scr}}^{\text{left}}$ and $\mathcal{KB}_{\text{scr}}^{\text{right}}$ are created (each according to the algorithm described in Section 4.2). Each of these HKBs represents the learned knowledge about the score change, given the orientation of the agent and a surrounding object focussing only on *one* of the objects currently above, below, left, or right of the agent. Every of the four smaller HKBs is created from an only four-dimensional matrix which contains the learned relative frequencies how often an action leads to a certain score change, given the agent's orientation and the type of one of the objects above, below, left or right of the agent, respectively. In case of $\mathcal{KB}_{\text{scr}}^{\text{above}}$, this matrix has the form $\hat{Q}_{\text{scr}}^{\text{above}} = (q_{s_{\text{above}}, s_{\text{ori}}, a, s_{\text{scr}}})$ with $s_{\text{above}} \in \mathbb{S}_{\text{obj}}$, $s_{\text{ori}} \in \mathbb{S}_{\text{ori}}$, $a \in \mathbb{A}$ and $s_{\text{scr}} \in \mathbb{S}_{\text{scr}}$, where $\mathbb{S}_{\text{obj}}$ is the set of object types, $\mathbb{S}_{\text{ori}}$ is the set of the agent's orientations, $\mathbb{A}$ is the agent's action space and $\mathbb{S}_{\text{scr}}$ is the set of score changes.

As the second step, the four smaller HKBs that have been created before are *merged* to the final HKB $\mathcal{KB}_{\text{scr}}$ by adding all rules of one level to the respective level in the merged HKB $\mathcal{KB}_{\text{scr}}$. If a rule with the same premise and the same conclusion already exists on this level, then the rule with the smallest weight is kept.

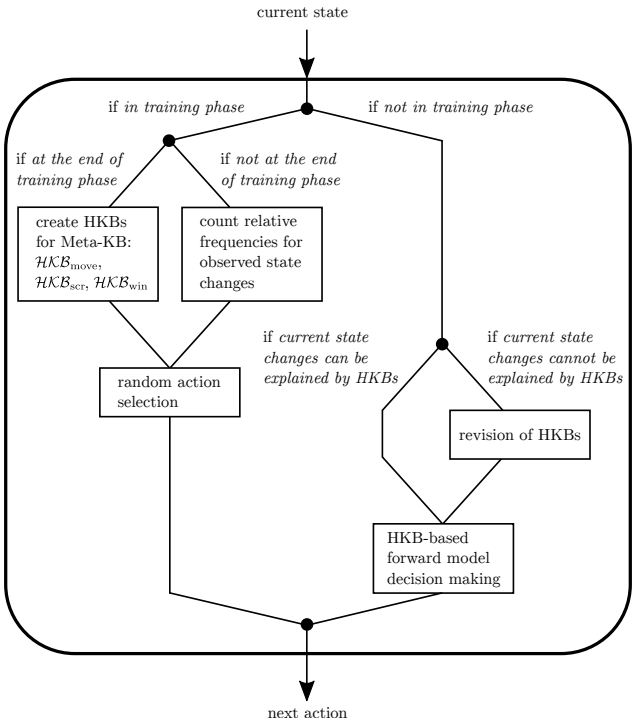


Fig. 3. Overview of the Agent Model Incorporating Learning, Revision, and Action Selection [6]

By dividing every HKB in several smaller HKBs (each representing only a subset of the state dimensions of the corresponding complete HKB), the computational challenge of creating the entire meta knowledge base could be overcome.

5.2 Action Selection

After a short learning phase we use the extracted HKBs as approximated forward model to predict the future states after picking action a. The action selection process will be guided by two sub-systems. MCTS is used for a broad exploration of longer action-sequences. In case no preferable solution can be found, we use BFS to find the shortest path to a state, which would yield an increase in points. In the following we will shortly review MCTS. The combination of both sub-systems and the adaptations made will be introduced in Section 5.

5.2.1 Monte Carlo Tree Search (MCTS)

In this study we will make use of MCTS for sampling long-term action sequences. MCTS is a heuristic search algorithm, which consists of four phases, (1) node selection, (2) node expansion, (3) simulation, and (4) backpropagation of the (expected) reward. The first two steps form the tree policy, while the latter two are also known as the default policy. The simulation during phase (3) consists of multiple rollouts of action sequences, which are simulated using a forward model. A rollout starts at the agent's current state and repeatedly chooses actions till either the end of an episode, at which the winner of the game is known, or any mid-game state is reached. In case the simulation is stopped before the end of an episode a scoring function is used to evaluate the value of the final state. The result of an action is estimated using a forward model, which describes the transition from state s to the next

state s' after using action a. The observed score at the end of an episode is propagated back to improve iteratively the value estimation of intermediate states. After all simulations are completed the agent uses its value estimate to choose and execute the action with the highest expected return. Note, that this search process is affected by the reward distribution and may fail in case of sparse rewards.

Multiple factors influence the capabilities of this search strategy. Next to the quanity and depth of a rollout, previous studies on card games showed that the quality of a rollout is a critical factor for a strong playing behavior [35], [36]. This introduces a trade-off between the depth, quantity and quality of a rollout during the simulation phase. All three will be limited during this study due to the short time span for action selection and the computational overhead of the approximated forward model. The additional time spent on the calculation of future states limits the search depth considerably. During our tests we aimed for 20 simulations using a depth of 20 actions. Using this approximation we are able to partially simulate future states. We use a discounted return based on all received simulated rewards between start and end of the rollout. The action, which yields the highest average return is chosen as the agent's next action.

Similar to the agent Yolobot we tested BFS as an alternative to MCTS. However, some adaptations to vanilla breadth first search were necessary to compensate for the additional time spent on calculating future states and the high number of states to be processed in deeper layers. For a fair comparison, we also added them to our implementation of MCTS.

- **Fast Forward Prediction/Macro Actions:** Many games consist of continuous movements in which the agent needs to use the same action multiple times in a row. We make use of this fact by multiplying the outcome of a relative movement times the block size of the game. This considerably reduces the simulation steps needed for long movements and allows the agent to explore far positions during a single rollout.
- **State Pruning:** In general, we cannot be sure which states to prune, since state transitions are only partially simulated. As an unpruned tree grows exponentially in size, it is near to impossible to reach higher search depths using breadth first search in the available time. Therefore, we consider predicted states to be equal in case they yield the same agent position, score, and winner. Using this pruning strategy movements such as "first up, then left" and "first left, then up" are considered to yield the same result and are processed only once.

5.3 Higher Level Summary of the Agent's Process

Finally, to complete this section, a brief higher level summary of the main steps of the agent's process will be provided here:

1) **Training Phase:**
   1.1 Random exploration to collect sensory data.
   1.2 Create HKBs as forward models from collected data.
2) **Evaluation Phase:**
   2.1 Perform action selection by MCTS/BFS based on the forward model from Step 1.2.
   2.2 If an observation is not consistent with the forward model from Step 1.2: Perform belief revision.

6 Evaluation

The proposed approach is evaluated in two phases. First, parameters of BFS and MCTS are tuned according to their resulting game-playing performance in terms of games of the GVGAI framework (Section 6.2). In the second part of our evaluation, the best performing parameter combination per algorithm will join a hypothetical competition of the GVGAI single-player learning track. Both evaluations will use the same evaluation setup, which is explained in Section 6.1 and resembles the competition's learning track rules of 2017. As the limits on training time were removed from the track's rule set in 2018, the agents of the subsequent years were developed under significantly different conditions and will therefore not be considered in the following evaluation.

6.1 Evaluation Setup

During both of the following evaluations, we will make use of the 10 games provided in the competition's training set 1. As suggested by the authors of the GVGAI framework, this set of training games has been chosen to represent games with differing characteristics [37]. Games vary in their scoring system, the type of included NPCs, the use of a resource system, the number and types of termination conditions as well as the number of actions available to the agent. It is the only set for which the results of submitted learning-track agents are publicly available. A comparison to agents of the game-playing track is not considered in the context of this paper since the track's underlying learning conditions are vastly different to the learning track.

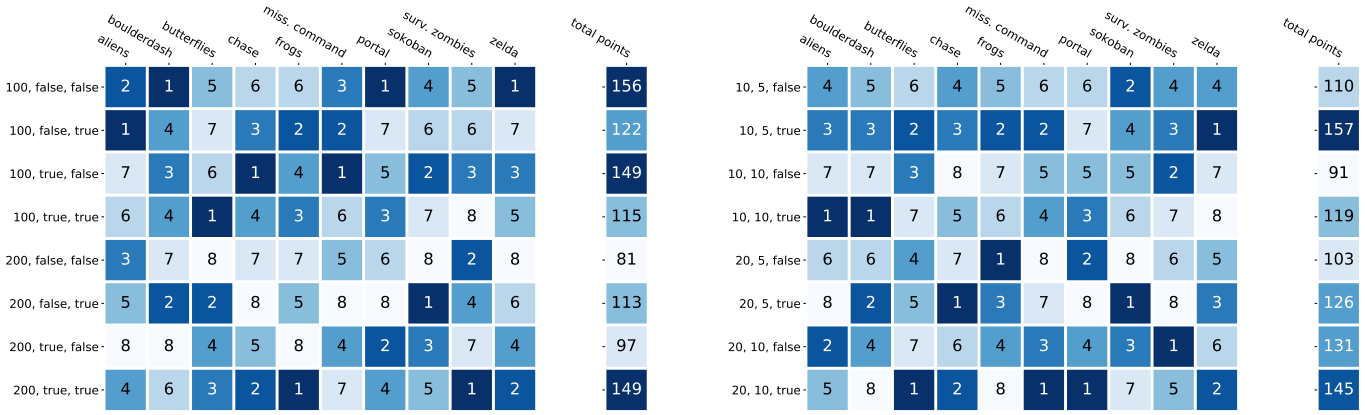
For each of these 10 games the agents' are trained for 5 minutes by playing 3 test levels. During the training, the agent is required to play each training level at least once. After this initial training phase, the agents game-playing performance is tested using 2 previously unseen levels. Agents are later compared according to their average win-rate, average score, average ticks until a game has been won, and average ticks until a game has been lost. Based on these values we determine a ranking of the agents for each of the 10 games. Finally, we apply the Formula-1 scoring system and rank the agents according to their sum of points achieved through all games.[3]

During the training, our proposed agent uses the first 50 seconds to repeatedly play through provided training levels. The remaining training time is used to process observed interactions and construct the agent's knowledge bases. During the evaluation, the approximated forward model is used to play each test level 20 time to measure the agent's game playing performance.

6.2 Parameter Optimization

For optimising the agents' parameters we performed a grid search during which each combination was tested by playing the 10 training set games. Results of this process are summarised in Figure 4 which shows the agent's ranking per game and their total score. The influence of each parameter on the agent's average win-rate, score, and ticks per game is shown in Figures 5 and 6. For the latter we concentrate

3. Depending on their ranking the agents receive 25, 18, 15, 12, 10, 8, or 6 points, whereas the best agent receives the most points.



(a) BFS Agents (expansions, macro actions, state pruning)

(b) MCTS Agent (nr of rollouts, rollout depth, macro actions)

Fig. 4. Agent rankings per game for different parameterizations. Total score per agent when applying the Formula-1 scoring scheme.

on games that have been lost since the variation of the game length is larger than in games that have been won.

For BFS-based agents we adjusted the number of node expansions (100 or 200), the usage of macro actions, and the usage of state pruning, resulting in 8 combinations. Results of the BFS agent optimisation, show that a change of a single parameter often only minorly influences the agent's results. The agent's total points show that the BFS agent using 100 node expansions without using macro actions or state expansions results in the best performance.

For the MCTS-based agents we adjusted the number of tree expansions (5 or 10), the rollout length (10 or 20), and the usage of macro actions during the simulation, also resulting in 8 combinations. Due to the competition's tough time-limits for action selection (40ms) and the additional computational cost of the approximated forward model, the search is reduced to a small number of tree expansions and comparably short rollout lengths. The MCTS parameter optimisation showed that changing a single parameter often has not much of an effect on the agents game-playing performance. The main difference in the agents' win-rate can be found when comparing MCTS agents using short rollouts with agents using longer rollouts. Especially in games with a large number of random events, such as butterflies, shorter rollouts seems to be beneficial since the accuracy of made predictions will lessen with increasing length of the rollout. Overall, the agent using 10 tree expansions, a simulation depth of 5, and making use of macro actions during the simulation performed best.

6.3 Comparison to Agents of the 2017's Learning Track

Subsequently, we compared the best-performing MCTS and BFS agents with agents of the 2017's learning track. Since the sourcecode of these agent's is unavailable at time of writing this paper, the evaluation is based on reported results on the competitions website[4]. Unfortunately, the number of ticks until a win or loss cannot be used as additional criteria, since the competition website does not list them. In case the win-rate and score are not sufficient to determine a ranking among a set of agents we consider it a draw.

Table 1 lists the agents' win-rate and average score for each of the 10 games. The number of points per game and the total score across all games are listed in Table 2. Our results show that the proposed agent model using the MCTS algorithm is able to overall score the most points (203). Given a margin of 34 points, the agent by Ilhan and Etaner-Uyar [12] performed second best with a total of 169 points. Similarly, the BFS agent performed well on average, ranking third when being compared with other agents.

6.4 Discussion of the Results

The MCTS-based agent is the single best agent in 5 out of 10 games. The proposed agent model performed better than other agents in the games *Boulderdash*, *Butterflies*, and *Survive Zombies*. In those games the agent wins by moving on the same or neighbouring position as other objects or characters. Here, the applied search scheme can work to its full potential, since extracted knowledge bases consider neighboring relations as important criteria for the prediction of future states. After the agent learns how to move (represented in $\mathcal{KB}_{\text{move}}$), the agent easily scores points by searching a path to the closest objective. In the game *Survive Zombies* the agent also needs to apply knowledge about the winning and losing condition ($\mathcal{KB}_{\text{win}}$). In case the player runs out of health he quickly needs to collect a healing item while avoiding zombies. As it was the case for chasing, the same search schemes are efficient in avoiding sources of danger while searching for an item.

In two of the remaining games (*Frogs*, *Portals*) our agent and all other agents score zero points. A closer inspection of extracted knowledge bases shows that in these games the agent failed to learn a prediction of upcoming rewards due to the small number of positive training examples in these sparse reward games. For this reason, actions cannot be differentiated according to their expected return, which results in a random behaviour. The same effect can be observed in the game *Sokoban*, which additionally to the sparse reward requires planning of a long action sequence.

4. http://gvgai.net/gvg_rankings_learning_1p.php

Currently learned knowledge bases do not consider movement of non-player entities, which are necessary to solve the puzzles provided in each level. For this reason it is impossible to plan the necessary action sequences to win the game. Hence, the agent once again falls back to choosing actions at random.

Despite the good performance, our agent has still much more room for improvement in games such as *Aliens* and *Boulderdash*. The current score could be improved by taking the actions of non-player characters into account. For example, in the game *Alien* the movement of the aliens and the fired shots can be predicted very easily. Using this information should allow to plan how an enemy can be hit.

7 Conclusion and Future Work

Our proposed algorithm learns a prediction of future states based on a given state and an action to be applied. This prediction model is split into multiple individual sub-models, which are first trained on sample interactions with the game and later revised with an AGM conforming belief revision approach, in case of contradictory observations. Using the extracted model we apply MCTS and BFS to find the best possible action at each game tick. The used search procedures were optimized by applying a state pruning, which reduces the number of evaluated states during the search phase.

We evaluated our approach in 10 games of the GVGAI competition. The proposed Forward Model Approximation agent has resulted in a better average game-playing performance than other game learning agents across a total of 10 games of the GVGAI framework. Despite using only one minute of training time, the agent has shown capable in playing some of the tested games much better than other agents (cf. Boulderdash, Butterflies, and Survive Zombies).

Our evaluation shows the potential of forward model approximation. However, there is still much room for improvement, since the current version only considers the agents movement during future states. Complex interaction with other game elements are not yet modeled. Further improvements could be achieved by generalizing the prediction to many other attributes. Future work could focus on analyzing the capabilities of forward model approximation by increasing the number of predicted variables, while keeping the computational expense as low as possible.

Additional project files and the detailed results of our evaluation can be found at:
https://github.com/ADockhorn/Forward-Model-Approximation

References

[1] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "General video game ai: Competition, challenges and opportunities," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence and the Twenty-Eighth Innovative Applications of Artificial Intelligence Conference*, D. Schuurmans and M. Wellman, Eds. Palo Alto, California: AAAI Press, 2016, pp. 4335–4337.

[2] C. B. Browne, E. Powley, D. Whitehouse, S. M. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, and S. Colton, "A Survey of Monte Carlo Tree Search Methods," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 4, no. 1, pp. 1–43, mar 2012.

[3] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open Loop Search for General Video Game Playing," *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*, pp. 337–344, 2015.

[4] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of Vanilla Rolling Horizon Evolution Parameters in General Video Game Playing," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2017, vol. 10199 LNCS, pp. 418–434.

[5] R. D. Gaina, S. M. Lucas, and D. Perez-Liebana, "Rolling horizon evolution enhancements in general video game playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, aug 2017, pp. 88–95.

[6] A. Dockhorn and D. Apeldoorn, "Forward Model Approximation for General Video Game Learning," in *Proceedings of the 2018 IEEE Conference on Computational Intelligence and Games (CIG'18)*. IEEE, aug 2018, pp. 425–432.

[7] M. Genesereth, N. Love, and B. Pell, "General Game Playing: Overview of the AAAI Competition," *AI Magazin*, vol. 26, no. 2, pp. 62–72, 2005.

[8] T. Schaul, "A video game description language for model-based or interactive learning," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2013.

[9] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, and S. M. Lucas, "Analyzing the robustness of general video game playing agents," *IEEE Conference on Computatonal Intelligence and Games, CIG*, 2017.

[10] R. D. Gaina, J. Liu, S. M. Lucas, and D. Pérez-Liébana, "Analysis of vanilla rolling Horizon evolution parameters in general video game playing," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, vol. 10199 LNCS, pp. 418–434, 2017.

[11] V. Mnih, K. Kavukcuoglu, D. Silver, A. a. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, S. Petersen, C. Beattie, A. Sadik, I. Antonoglou, H. King, D. Kumaran, D. Wierstra, S. Legg, and D. Hassabis, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, pp. 529–533, feb 2015.

[12] E. İlhan and A. S. Etaner-Uyar, "Monte Carlo tree search with temporal-difference learning for general video game playing," in *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. New York: IEEE, aug 2017, pp. 317–324.

[13] H. van Seijen, A. R. Mahmood, P. M. Pilarski, M. C. Machado, and R. S. Sutton, "True Online Temporal-Difference Learning," *Journal of Machine Learning Research (JMLR)*, vol. 17, pp. 1–40, dec 2015.

[14] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv: . . .* , pp. 1–9, dec 2013.

[15] V. Mnih, A. P. Badia, M. Mirza, A. Graves, T. P. Lillicrap, T. Harley, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *CoRR*, vol. abs/1602.01783, 2016.

[16] S. Racanière, T. Weber, D. P. Reichert, L. Buesing, A. Guez, D. Rezende, A. P. Badia, O. Vinyals, N. Heess, Y. Li, R. Pascanu, P. Battaglia, D. Hassabis, D. Silver, and D. Wierstra, "Imagination-augmented agents for deep reinforcement learning," *Advances in Neural Information Processing Systems*, vol. 2017-December, no. Nips, pp. 5691–5702, 2017.

[17] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to control: Learning behaviors by latent imagination," *arXiv preprint arXiv:1912.01603*, 2019.

[18] D. Ha and J. Schmidhuber, "Recurrent world models facilitate policy evolution," in *Advances in Neural Information Processing Systems 31*. Curran Associates, Inc., 2018, pp. 2451–2463.

[19] A. Braylan and R. Miikkulainen, "Object-Model Transfer in the General Video Game Domain," *Aiide*, pp. 136–142, 2016.

[20] M. Guzdial, B. Li, and M. O. Riedl, "Game engine learning from video," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 3707–3713, 2017.

[21] A. Dockhorn and R. Kruse, "Detecting Sensor Dependencies for Building Complementary Model Ensembles," in *Proceedings. 28. Workshop Computational Intelligence, Dortmund, 29.-30. November 2018*, 2018.

[22] A. Dockhorn, S. M. Lucas, V. Volz, I. Bravi, R. D. Gaina, and D. Perez-Liebana, "Learning Local Forward Models on Unforgiving Games," in *IEEE Conference on Games (COG)*. London: IEEE, aug 2019, pp. 1–4.

[23] S. M. Lucas, A. Dockhorn, V. Volz, C. Bamford, R. D. Gaina, I. Bravi, D. Perez-Liebana, S. Mostaghim, and R. Kruse, "A Local Approach to Forward Model Learning: Results on the Game of Life Game," in *IEEE Conference on Games (COG)*. IEEE, aug 2019, pp. 1–8.

TABLE 1
Agent performances on the GVGAI-Learning Track Games,
G1 = Aliens, G2 = Boulderdash, G3 = Butterflies, G4 = Chase, G5 = Frogs,
G6 = Missile Command, G7 = Portals, G8 = Sokoban, G9 = Survive Zombies, G10 = Zelda
Agents: FMA = Forward Model Approximation, DUU = DontUnderestimateUchiha

Agent Name	Win Rate / Average Points per Game									
	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10
FMA-MCTS	0.25 / 32.6	**0.00 / 4.0**	**0.90 / 29.8**	**0.00 / 1.6**	**0.00 / 0.0**	0.10 / 0.3	**0.00 / 0.0**	0.05 / 0.5	**0.05 / 2.7**	**0.05 / 0.7**
FMA-BFS	0.30 / 35.0	0.00 / 1.3	0.85 / 23.3	0.00 / 0.9	**0.00 / 0.0**	0.05 / 0.3	**0.00 / 0.0**	0.05 / 0.6	0.00 / 1.4	0.05 / 0.6
ercumentilhan	0.45 / 37.3	0.00 / 1.3	0.90 / 18.6	0.00 / 0.7	**0.00 / 0.0**	0.50 / -0.1	**0.00 / 0.0**	0.00 / 0.7	0.00 / 0.1	0.00 / -0.1
sampleRandom	0.10 / 29.8	0.00 / 1.4	0.85 / 19.3	0.00 / 0.5	**0.00 / 0.0**	0.50 / -0.5	**0.00 / 0.0**	0.10 / 0.8	0.00 / -0.1	0.00 / 0.3
sampleLearner	0.20 / 34.5	0.00 / 1.3	0.40 / 15.3	0.00 / 0.4	**0.00 / 0.0**	0.50 / -0.4	**0.00 / 0.0**	0.00 / 0.6	0.00 / 0.1	0.00 / -0.3
DUU	**0.75 / 41.2**	0.00 / 0.3	0.15 / 11.6	0.00 / 0.0	**0.00 / 0.0**	0.50 / -0.5	**0.00 / 0.0**	0.00 / 0.0	0.00 / -0.1	0.00 / 0.0
kkunan	0.35 / 35.6	0.00 / 0.7	0.10 / 11.4	0.00 / 0.0	**0.00 / 0.0**	**0.50 / 0.4**	**0.00 / 0.0**	0.00 / 0.0	0.00 / -0.2	0.00 / -0.3
YOLOBOT	0.45 / 32.3	0.00 / -0.3	— / —	0.00 / 0.0	**0.00 / 0.0**	0.00 / 0.0	**0.00 / 0.0**	**0.10 / 1.0**	0.00 / 0.0	0.00 / -0.5

TABLE 2
Agent score using the Formula-1 score system based on an agent's average points per game

Agent Name	G1	G2	G3	G4	G5	G6	G7	G8	G9	G10	Total
FMA-MCTS	8	25	25	25	25	8	25	12	25	25	203
FMA-BFS	10	15	15	18	25	6	25	15	18	18	165
ercumentilhan	18	15	18	15	25	18	25	10	15	10	169
sampleRandom	4	18	12	13	25	12	25	18	10	15	152
sampleLearner	6	15	10	12	25	15	25	8	15	8	139
DontUnderestimateUchiha (DUU)	25	6	8	10	25	12	25	6	10	12	139
kkunan	12	8	6	10	25	25	25	4	6	8	129
YOLOBOT	15	4	4	10	25	4	25	25	4	4	120

[24] M. I. Jordan and R. A. Jacobs, "Learning to control an unstable system with forward modeling," in *Advances in Neural Information Processing Systems 2*, D. S. Touretzky, Ed. Morgan-Kaufmann, 1990, pp. 324–331.

[25] A. Dearden and Y. Demiris, "Learning forward models for robots," in *Proceedings of the 19th International Joint Conference on Artificial Intelligence*, ser. IJCAI'05. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2005, p. 1440–1445.

[26] D. Nguyen-Tuong and J. Peters, "Model learning for robot control: a survey," *Cognitive Processing*, vol. 12, no. 4, pp. 319–340, Apr. 2011. [Online]. Available: https://doi.org/10.1007/s10339-011-0404-1

[27] R. S. Sutton and A. G. Barto, *Reinforcement Learning*, 2nd ed. Cambridge: The MIT Press, 2018.

[28] G. N. Yannakakis and J. Togelius, *Artificial Intelligence and Games*. Springer, 2018, http://gameaibook.org.

[29] D. Apeldoorn and G. Kern-Isberner, "Towards an understanding of what is learned: Extracting multi-abstraction-level knowledge from learning agents," in *Proceedings of the Thirtieth International Florida Artificial Intelligence Research Society Conference*, V. Rus and Z. Markov, Eds. Palo Alto, California: AAAI Press, 2017, pp. 764–767.

[30] ——, "When should learning agents switch to explicit knowledge?" in *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, ser. EPiC Series in Computing, vol. 41. EasyChair Publications, 2016, pp. 174–186.

[31] ——, "An agent-based learning approach for finding and exploiting heuristics in unknown environments," in *Proceedings of the Thirteenth International Symposium on Commonsense Reasoning, London, UK, November 6-8, 2017*, ser. CEUR Workshop Proceedings, A. S. Gordon, R. Miller, and G. Turán, Eds., vol. 2052. Aachen: CEUR-WS.org, 2018.

[32] D. Apeldoorn, "InteKRator toolbox for integrating machine learning with knowledge representation and belief revision," 2020. [Online]. Available: https://gitlab.com/dapel1/intekrator_toolbox

[33] C. E. Alchourrón, P. Gärdenfors, and D. Makinson, "On the logic of theory change: Partial meet contraction and revision functions," *Journal of Symbolic Logic*, vol. 50, no. 2, p. 510–530, 1985.

[34] S. O. Hansson, "Logic of belief revision," in *The Stanford Encyclopedia of Philosophy (Winter 2017 Edition)*, E. N. Zalta, Ed. Metaphysics Research Lab, Stanford University, 2017.

[35] A. Dockhorn, C. Doell, M. Hewelt, and R. Kruse, "A decision heuristic for Monte Carlo tree search doppelkopf agents," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, nov 2017, pp. 1–8.

[36] A. Dockhorn, T. Tippelt, and R. Kruse, "Model Decomposition for Forward Model Approximation," in *2018 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, nov 2018, pp. 1751–1757.

[37] D. Perez-Liebana, S. Samothrakis, J. Togelius, T. Schaul, S. M. Lucas, A. Couetoux, J. Lee, C.-U. Lim, and T. Thompson, "The 2014 general video game playing competition," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. 8, no. 3, pp. 229–243, Sep. 2016.

Daan Apeldoorn primarily works for the Institute for Medical Biostatistics, Epidemiology and Informatics (IMBEI) in the Medical Informatics department at the University Medical Centre of the Johannes Gutenberg University Mainz, Germany, and additionally for the Z Quadrat GmbH in Mainz. He is a PhD student at the TU Dortmund University. His research focuses on the incorporation of symbolic and sub-symbolic AI approaches—especially by extracting and (re)integrating symbolic knowledge in the context of learning agents. He is also active in the field of multi-agent systems with application in (hospital) logistics. In the past, he worked as a scientific staff member for the TU Dortmund University and the University of Koblenz-Landau.

Alexander Dockhorn is PhD student at the Otto von Guericke University in Magdeburg. In his research, he combines computational intelligence in games and intelligent data analysis with the focus on partial information games and the active learning of rules and strategies from successful playthroughs. He is student member of the IEEE and its Computational Intelligence Society and serves as chair of the IEEE CIS Competitions Sub-Committee.

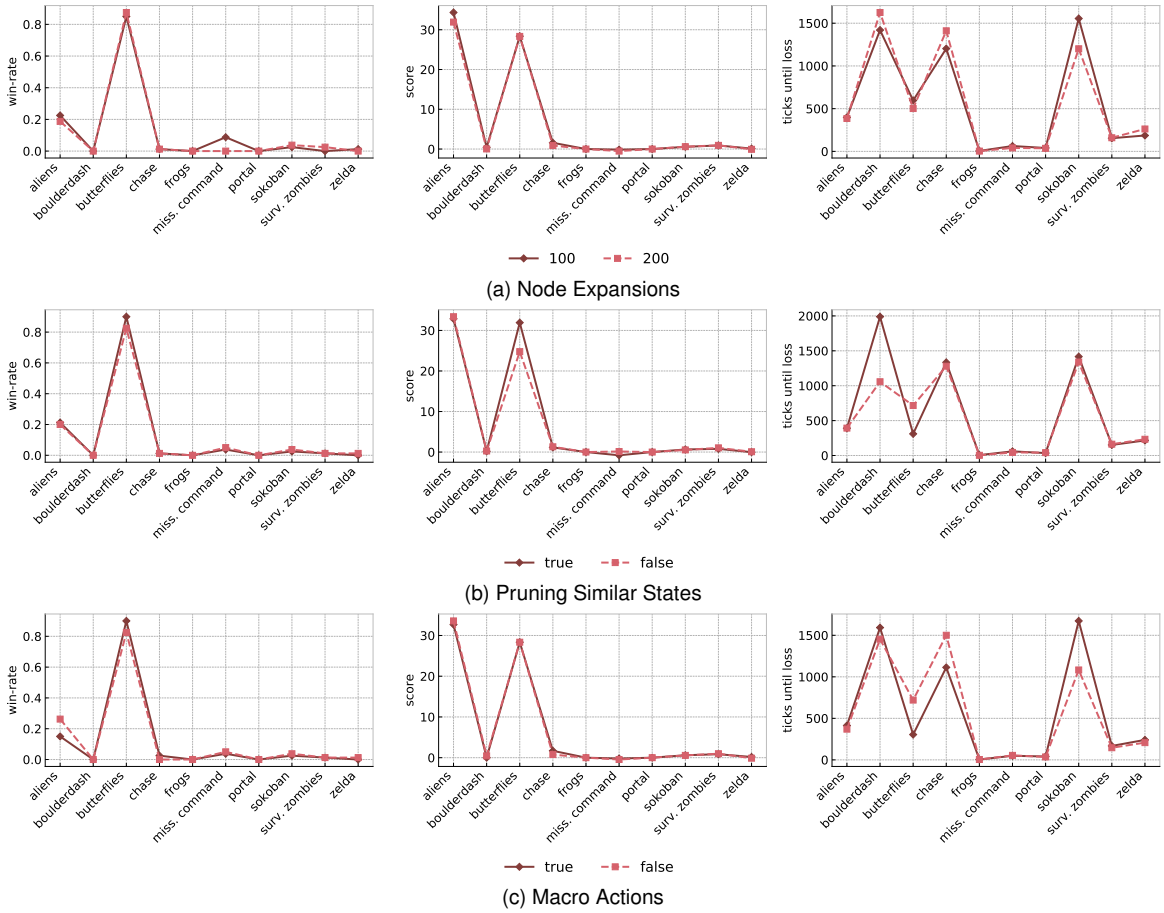


Fig. 5. Comparison of BFS Agent Parameterisations

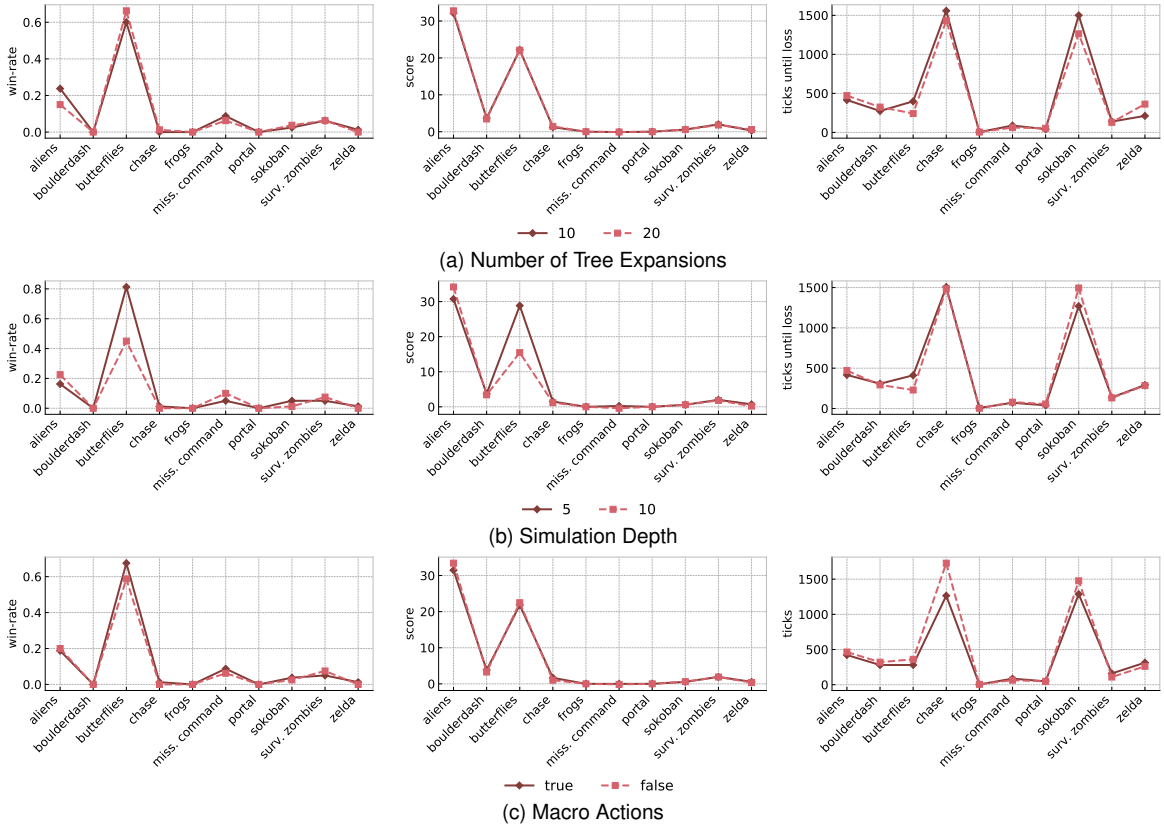


Fig. 6. Comparison of MCTS Agent Parameterisations