Forward Model Approximation for General Video Game Learning

Alexander Dockhorn
Otto von Guericke University
Magdeburg, Germany
alexander.dockhorn@ovgu.de

Daan Apeldoorn
Z Quadrat GmbH
Mainz, Germany
daan.apeldoorn@z-quadrat-mainz.de

***Abstract*—This paper proposes a novel learning agent model for a General Video Game Playing agent. Our agent learns an approximation of the forward model from repeatedly playing a game and subsequently adapting its behavior to previously unseen levels. To achieve this, it first learns the game mechanics through machine learning techniques and then extracts rule-based symbolic knowledge on different levels of abstraction. When being confronted with new levels of a game, the agent is able to revise its knowledge by a novel belief revision approach. Using methods such as Monte Carlo Tree Search and Breadth First Search, it searches for the best possible action using simulated game episodes. Those simulations are only possible due to reasoning about future states using the extracted rule-based knowledge from random episodes during the learning phase. The developed agent outperforms previous agents by a large margin, while still being limited in its prediction capabilities. The proposed forward model approximation opens a new class of solutions in the context of General Video Game Playing, which do not try to learn a value function, but try to increase their accuracy in modelling the game.**

***Index Terms*—Forward Model Approximation, General Video Games, exception-tolerant Hierarchical Knowledge Bases, Belief Revision, Monte Carlo Tree Search, Breadth First Search**

I. Introduction

The development of General Intelligence (i. e., intelligent algorithms that are able to cope with multiple kinds of different problems) is one of the key long term goals in Artificial Intelligence (AI) research. To evaluate approaches for this long term goal, in recent years, the General Game Playing challenge [1] and (more recently) the General Video Game Playing Artificial Intelligence (GVGAI) competition [2] have been developed, where agents have to play several different types of games which are not (concretely) known in advance.

In this paper, we present a novel approach of a learning agent which learns to play different video games in the context of the GVGAI competition [2]. To achieve this, we combine multiple concepts from both the symbolic and the subsymbolic artificial intelligence communities using machine learning together with belief revision and action selection. More precisely, our agent first uses statistical methods to learn the game mechanics in form of an *approximated forward model*. In this study we use *exception-tolerant Hierarchical Knowledge Bases* (HKBs) [3], containing rules on multiple levels of abstraction, to learn and store the approximated forward model. These knowledge bases are later revised when the agent is confronted with new or changed environments (e. g., new levels) by a novel belief revision approach for HKBs.

Those knowledge bases can be used while playing the game for reasoning about the value of future states. Based on previous experiences we can partially simulate the outcome of our future actions, enabling us to use well-known search schemes for action selection. In this study we use Monte Carlo Tree Search (MCTS) and Breadth First Search (BFS) to evaluate future states. Finally, we choose the action with the highest expected outcome and return it for execution.

The main contributions of the paper are:

- a learning approach which is able to quickly learn the mechanics of dynamic environments (e. g., games) in form of an approximated forward model
- representing and storing the approximated forward model in human readable format as HKBs (i. e., rule-based knowledge which is organized on multiple levels of abstraction)
- a simple belief revision approach for HKBs
- an agent architecture using the generated approximated forward model with a flexible action selection mechanism
- an agent model to be used for the GVGAI learning-track

Section II briefly outlines related work and in Section III the preliminaries needed for our agent model will be introduced. After that, in Section IV, we describe the details of creating and using an approximated forward model in context of an agent for the GVGAI competition. In Section V, we evaluate and compare the proposed approach with other known agents based on several games from the GVGAI competition. A conclusion and an outlook on future work are given in Section VI.

II. General Game Playing

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 [1] asked its participants to implement agents, which can compete in a set of previously unknown games. Playing 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 [4] was created. Based on his work the General Video Game AI (GVGAI) competition was created. It offers a framework, in which many arcade like video games

were replicated. 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 buttons "Up", "Down", "Right", "Left", "Action". However, the outcome of each action 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. In the remainder of this paper we will focus on the study of the learning track.

A. 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.

In 2017 a total of 22 submissions entered the competition (including 5 sample submission provided by the competition hosts). The currently best performing agent, Yolobot [5], 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 [6]). 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 [5].

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 [7] 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.

B. GVGAI - Single Player Game Learning Track

In contrast to the game playing track, the game learning track has an increased difficulty due to the forward model being removed as a source of information from the agent. All previously discussed methods heavily rely on such a forward model, which enables the agent to run simulations for all possible actions. Therefore, new algorithms need to be found for solving problems of this domain.

The agent developed by Ercüment İlhan [8] uses an MCTS agent, which was enhanced with an online on-policy temporal-difference learning method, called true online Sarsa(λ) [9]. 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 only placed 5th in the evaluation game set (slightly ahead of an adaptation of the Yolobot from the game playing track). Not much information is available on the other agents.

Surprisingly the random controller provided by the competition organizers performed second best in the evaluation set, just three points worse than the first place. 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 games rules ($\leq$ 5 minutes) and decision-making (40ms) drastically limit the applicability of iterative methods or long search procedures. Hence, there is still more room for improvement.

Our work is motivated by the huge performance gap between agents of the game playing and game learning track and the idea that machine learning combined with methods from symbolic knowledge representation can contribute 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.

III. Preliminaries

This section will briefly outline the preliminaries and components needed for our agent model introduced in Section IV for the GVGAI learning track. We will first explain the basics needed for our novel *approximated forward model* for which it is useful to define Hierarchicle Knowledge Bases (HKB) and how they can be learned (Subsections III-A to III-C). These sections will closely follow several preliminary works, especially [3], [10], [11]. After that, the reasoning algorithm defined on HKBs will be briefly summarized, following [11] (Subsection III-D). Subsequently, we will introduce some *modifications* on HKBs and the learning process (Subsection III-E) to fit the needs of our agent model. Finally, we will propose a belief revision approach for HKBs (Section III-F) and describe a module for action selection (Subsection III-G).

A. Basic Agent Model for HKBs

We consider an agent which is equipped with n sensors through which it can perceive its current state in the environment (e. g., a game) and which is able to perform actions from a predefined action space (e. g., the keys to be pressed on the controller). Furthermore, the agent can perceive whether the performed actions were good, 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, 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}. In general a game can be modelled as a probability distribution over $P(s_{t+1}, r_{t+1}|s_0, a_0, r_1, s_1, a_1, \ldots, r_t, s_t, a_t)$, which maps the probability of each successor state and its accompanied reward depending on all previous interactions. Analysing or storing such a probability distribution is near to impossible, due to its exponentially growing complexity. When the process to be analysed fulfils the Markov Property, the probability distribution reduces to $P(s_{t+1}, r_{t+1}|s_t, a_t)$, in which each successor state and its reward is only dependent on the environment's last state and the agent's last chosen action. 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 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 is the case a complete reconstruction of the state $\mathbb{S}$ can be achieved by modelling each independent component separately.

The following subsections introduce HKBs, which will be used for learning and revising such classifiers during continuous interactions with the games environment.

B. Definition of HKBs

In [10], an extraction approach is proposed which is able to extract an HKB from a weighted state-action pair representation (where the maximum weight determines the best action, given a state). This section only provides the main definitions needed to understand the basic idea of HKBs and how they can be learned from weighted state-action pairs by closely following [3], [10], [11]. For more details on HKBs, the reader should refer to the original literature mentioned here. 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 (in contrast to *Exception Lists* [12]). According to [11] 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 $s := s_1 \wedge \ldots \wedge s_n$ of all values s_i currently perceived by an 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 $S \subset \{s_1, \ldots, s_n\}$ 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 an 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, *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 $\mathcal{KB} := \{R_1, \ldots, R_{n+1}\}$ of $n+1$ rule sets, where n is the number of sensors (i. e., the number of state space dimensions). Every set $R_{i<n+1}$ 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 $\rho \in R_i$ of length $|S_\rho| = i - 1$.*

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. [3]):

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 $S_\upsilon \subset S_\rho$, $a_\upsilon = a_\rho$ and $w_\upsilon > w_\tau$.*

C. 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 originally introduced in [10], closely following [11] here:

The approach takes a weighted state-action pair representation $\hat{Q}$ as input and returns an HKB $\mathcal{KB}^{\hat{Q}}$ which reflects the knowledge contained in $\hat{Q}$ by performing the following steps:

1) *Initial creation of rule sets:*
   In the first step, the multiple abstraction levels $R_1, \ldots, R_{n+1}$ 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.[1]
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 $S_\sigma \subset S_\rho = \{s_1, \ldots, s_{j-1}\}$ 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

[1] For performance reasons, only state-action pairs can be considered here that contribute to the best policy found by the preceding learning process (see [3] for a first attempt to a more efficient algorithm that preselects potentially relevant rules in this step).

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.

D. Reasoning on HKBs

This section briefly summarizes the basic idea of the efficient reasoning algorithm on HKBs which was first introduced in [10]. The summary closely follows [11]:

Given perceived sensor values $s_1, ..., s_n$, the reasoning algorithm searches an HKB upwards (starting from the bottom-most level R_{n+1}) for the first rule, which premise is fulfilled. This rule is then returned as concluding action (see [10] 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 serves as a heuristic), in case no more specific rule with a fitting premise could be found.

E. Modifications for Our Agent Model

In this section, the original HKB approach according to [3], [10], [11] (which was outlined in the previous Sections III-A to III-C) will be modified to fit the needs of our agent model for the learning track of the GVGAI competition. 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 (which can also be exploited during a learning process as has been demonstrated in [10], [11]). Nevertheless, according to the 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 Rules/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 one (or more) sensor value(s) of a subsequent state (resulting from action a performed in state p_ρ) and $w_\rho \in [0, 1]$ is the rule's weight. The sensor values of p'_ρ do not necessarily need to be of the same sensors used for p_ρ.*

Note that since the creation of HKBs from data is computationally rather expensive (cf. [10], see [3] for a first attempt towards a faster algorithm for HKB extraction). 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 IV.)

In addition, the extraction algorithm described in Section III-C will be extended by the following filter at the end of Step 5: All rules $\rho \in R_{j>1}$, which premise does not contain an action will be removed.

In the following, the idea of such modified HKBs will initially be explained in the context of the game *Butterflies* from the GVGAI competition framework [2].

Example 1 (Butterflies) *We consider the game Butterflies from the GVGAI competition framework [2], where an agent has to collect butterflies by touching them (see left part of Figure 1): Every time when collecting a butterfly, the agent's current score is increased by 2. To learn knowledge about the scoring of the game, the agent's surrounding objects and its orientation are considered as a subset of the state space dimensions. Furthermore, the agent's action space is given as usual (i. e., $\mathbb{A} := \{\text{Up}, \text{Down}, \text{Left}, \text{Right}, \text{Use}\,(, \text{None})\}$). After a short training phase, the learned HKB regarding the knowledge of the scoring of the game (i. e., which actions lead to which changes regarding the agent's current score, given a state) is shown in the right part of Figure 1.*

$\mathcal{KB}_1$:

score±0 [0.944]	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

Fig. 1. Excerpt from the First Level of the GVGAI Game *Butterflies* with Corresponding HKB of the Agent's Knowledge About the Scoring in the Game After a Short Training Phase

The HKB resulting from Example 1 (right part of Figure 1) can be read as follows: According to the single rule score±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, 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.

F. Belief Revision on 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 introduces a novel belief revision approach for HKBs: The proposed algorithm is a first simple attempt to realize revision on HKBs and leaves a lot of room for improvements. Nevertheless, it allows the agent to revise the learned knowledge represented by an HKB, in case the environment changes.

We consider a learned HKB with the modifications described in Section III-E and the reasoning algorithm described in Section III-D: Given a state representation s and an action a, and given that the representation of a subsequent state s' inferred through the HKB is *inconsistent* with the corresponding representation of the actual perceived subsequent state s'_{per} of the agent (i. e., $s' \not\equiv s'_{\text{per}}$), the basic processing of the revision algorithm is as follows:

- *Adding a new exception:*
  If the rule leading to the inconsistent inference is located on a level $R_{j<n+1}$, then a new rule is added on level R_{n+1} with the updated conclusion according to the actual perceived subsequent state.
- *Exchanging an existing exception:*
  Otherwise, if the inconsistent inference is located on the most specific level R_{n+1}, then the conclusion of that rule is simply replaced by the correct conclusion according to the actual perceived subsequent state.

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.

G. 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 IV.

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 backpropagated to iteratively improve 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.

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 [13], [14]. 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. Subsection IV-B will discuss our optimizations for MCTS, which facilitate a higher quantity of rollouts.

IV. Agent model

The agent model builds on the preliminaries described in Section III. First, the basic composition of the concepts required to learn the game mechanics will be summarized (Section IV-A). Consequently, this section focuses on how the action selection is realized based on these ideas and which modifications where necessary to optimize the process (IV-B). An overview of the agent model is provided in Figure 2.

A. 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 III-E) 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.

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. The following three aspects are covered:

- $\mathcal{HKB}_{\text{move}}$*, Relative Movement*: The movement depending on the relative position to other objects (e. g., obstacles like "objects of this type cannot be passed").
- $\mathcal{HKB}_{\text{scr}}$*, Scoring*: Score changes depending on interactions with other objects (e. g., beneficial objects like "collecting objects of this type increases the score by X").
- $\mathcal{HKB}_{\text{win}}$*, 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.

2) Fast Creation of HKBs for the Different Aspects: As mentioned in Section III-C and Section III-E, 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 $\{\mathcal{HKB}_{\text{move}}, \mathcal{HKB}_{\text{scr}}, \mathcal{HKB}_{\text{win}}\}$.

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

$\mathcal{HKB}_{\text{scr}}$ 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 $\mathcal{HKB}_{\text{scr}}$ could be created (according to the algorithm described in Section III-C) 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 $\mathcal{HKB}_{\text{scr}}$ directly from the seven-dimensional matrix $\hat{Q}_{\text{scr}}$, as a first step, the four smaller HKBs $\mathcal{HKB}_{\text{scr}}^{\text{above}}$, $\mathcal{HKB}_{\text{scr}}^{\text{below}}$, $\mathcal{HKB}_{\text{scr}}^{\text{left}}$ and $\mathcal{HKB}_{\text{scr}}^{\text{right}}$ are created (each according to the algorithm described in Section III-C). 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 $\mathcal{HKB}_{\text{scr}}^{\text{above}}$, 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.

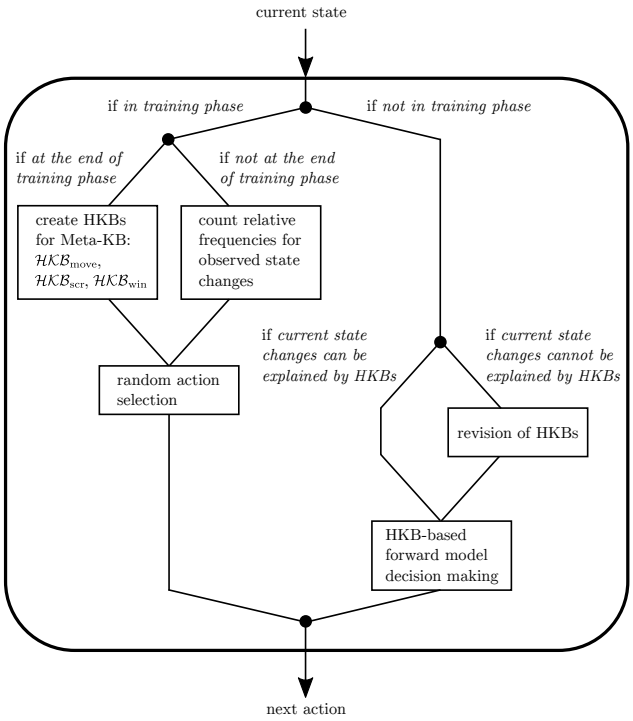

Fig. 2. Overview of the Agent Model Incorporating Learning, Revision, and Action Selection

As the second step, the four smaller HKBs that have been created before are *merged* to the final HKB $\mathcal{HKB}_{\text{scr}}$ by adding all rules of one level to the respective level in the merged HKB $\mathcal{HKB}_{\text{scr}}$. 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.

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 and all HKBs of the meta knowledge base can be created in acceptable time.

B. Action Selection

After the training time expired we use and revise the extracted HKBs during the action selection phase. Many agents in the game playing track already based their action selection on MCTS. Applying MCTS in the learning track is not straightforward, since the necessary simulation for the rollouts is not available. We use the generated HKBs to create an approximation of the 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 search 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 the future states and the high number of states to be processed in deeper layers. Since those methods showed to be beneficial in the application of both search procedures, we also added them to the MCTS implementation.

1) Fast Forward Prediction: 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.

2) 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 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 only processed once.

V. Evaluation

Our approach is evaluated and compared with other agents using multiple games from the GVGAI framework. In Subsection II-B we already summarized the previous agents' frameworks in learning and playing unknown games. Because the implementations of other agents are currently not public, we use the same set of games used in the CIG 2017 training set, for which the competition web page[2] offers detailed results on the agents' performances. For each of the 10 games we train our agent for about 50 seconds cycling through the first 3 levels, in contrast to a maximum of 5 minutes of training time provided by the competition framework. After the first 50 seconds we use the collected observation data to construct our knowledge bases, which takes less than 1 minute in which the agent continuous using random actions. Using the approximated forward model we play 10 rounds on each of the two remaining levels per game. The average score after playing those 20 rounds is reported in Table I. Next to the results of our agent, we list the performances of all the agents of the 2017 GVGAI learning-track competition.

This evaluation is limited due to the unavailability of agent implementations and the limited number of recorded results on the competition web page. Nevertheless, the 10 chosen games of the training set represent a well mixed set of games, which require the agents to play well in a variety of scenarios.

Our approach is the single best agent in 5 out of 10 games and second best in 1 game. We beat the previously best agent by a large degree in games such as *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, because the extracted knowledge predicts future states with high accuracy.

The proposed framework excels in games such as *Butterflies* and *Survive Zombies*. After the agent learns how to move (represented in $\mathcal{HKB}_{\mathrm{move}}$), he easily scores points by searching a path to the nearest objective, which is either a butterfly or a zombie respectively. In the game *Survive Zombies* he also needs to apply knowledge about the winning and losing condition ($\mathcal{HKB}_{\mathrm{win}}$). 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 the dangers while searching for an item.

In two of the remaining games (*Frogs*, *Portals*) our agent and all other agents score zero points. This is due to the sparse reward given in both games. For this reason, our agent is not able to rate an action based on the return, because the score does not change till the endpoint of our simulated episode. Hence, we are not able to differentiate between good and bad actions, such that our agent falls back to 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. 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.

To get an overview on the overall performance of our agent we applied the Formula 1 Scoring system, which was already used in the original competition. Here, the agents are ranked based on their average points per game. Depending on their ranking the agents receive 25, 18, 15, 12, 10, 8, or 6 points, whereas the best agent receives the most points. Table II shows the score each agent received per game and its sum over all games. Our results show that the proposed agent outperforms previous agents by a large margin of 32 points.

VI. Conclusion and Future Work

With the help of the proposed *Forward Model Approximation*, we are able to outperform other agents of the GVGAI learning track using only about a fifth of the available learning time, plus less than one additional minute for the creation of the knowledge bases. 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

[2]http://gvgai.net/gvg_rankings_learning_1p.php

TABLE I
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	0.30 / 35.4	**0.00 / 2.1**	**1.00 / 18.7**	**0.00 / 1.1**	**0.00 / 0.0**	0.50 / -1.8	**0.00 / 0.0**	0.10 / 0.9	**0.00 / 1.3**	**0.00 / 0.4**
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 II
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
Forward Model Approximation (FMA)	10	**25**	**25**	**25**	**25**	8	**25**	18	**25**	**25**	**211**
ercumentilhan	18	15	18	18	**25**	18	**25**	12	18	12	179
sampleRandom	6	18	15	15	**25**	12	**25**	15	10	18	159
sampleLearner	8	15	12	12	**25**	15	**25**	10	18	10	150
DontUnderestimateUchiha (DUU)	**25**	8	10	10	**25**	12	**25**	8	10	15	148
kkunan	12	10	8	10	**25**	**25**	**25**	8	6	10	139
YOLOBOT	15	6	6	10	**25**	6	**25**	**25**	12	6	136

sub-models, which are first trained on sample interactions with the game and later revised 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 developed agent overall outperforms previous algorithms. Our evaluation shows the enormous 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 evaluation can be found at: https://doi.org/10.17605/OSF.IO/VN3ZS

REFERENCES

[1] 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.

[2] 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.

[3] 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.

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

[5] 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.

[6] 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.

[7] 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.

[8] 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.

[9] 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.

[10] D. Apeldoorn and G. Kern-Isberner, "When should learning agents switch to explicit knowledge?" in *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, ser. EPiC Series in Computing, C. Benzmüller, G. Sutcliffe, and R. Rojas, Eds., vol. 41. EasyChair Publications, 2016, pp. 174–186.

[11] ——, "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.

[12] L. Michael, "Causal Learnability," in *Proceedings of the 22nd International Joint Conference on Artificial Intelligence (IJCAI 2011)*. Palo Alto, California: AAAI Press, 2011, pp. 1014–1020.

[13] 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. [Online]. Available: http://ieeexplore.ieee.org/document/8285181/

[14] A. Dockhorn, M. Frick, Ü. Akkaya, and R. Kruse, "Predicting Opponent Moves for Improving Hearthstone AI," in *17th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems (IPMU)*, 2018, p. (to be published).