Model Decomposition for Forward Model Approximation

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

Tim Tippelt
Intelligent Cooperative Systems
Otto von Guericke University
Magdeburg, Germany
tim.tippelt@ovgu.de

Rudolf Kruse
Intelligent Cooperative Systems
Otto von Guericke University
Magdeburg, Germany
rudolf.kruse@ovgu.de

***Abstract*—In this paper we propose a model decomposition architecture, which advances on our previous attempts of learning an approximated forward model for unknown games [1]. The developed model architecture is based on design constraints of the General Video Game Artificial Intelligence Competition and the Video Game Definition Language. Our agent first builds up a database of interactions with the game environment for each distinct component of a game. We further train a decision tree model for each of those independent components. For predicting a future state we query each model individually and aggregate the result. The developed model ensemble does not just predict known states with a high accuracy, but also adapts very well to previously unseen levels or situations. Future work will show how well the increased accuracy helps in playing an unknown game using simulation-based search algorithms such as Monte Carlo Tree Search.**

***Index Terms*—Forward Model Approximation, Decision Trees, Ensemble Learning, General Video Games**

I. Introduction

In recent years reinforcement learning showed to be one of the most successful methods in the context of computational intelligence in games [2], [3]. In a work of GoogleDeepMind an agent was trained to play a collection of "Atari 2600" games. The agent was often capable of playing on a super-human level, but a total of 38 days of simulated play time was used to learn each of the 49 games [4]. Another example is the work by OpenAI [2], in which they trained a team of agents playing the game Dota 2. While the trained agents are able to beat semi-pro teams in a restricted version of the game, the total time of simulated play exceeds thousands of years. Despite the impressive results, such long learning times are rather unreasonable when compared to a human player.

Recently, we proposed a different approach for learning unknown games from scratch [5]. In contrast to reinforcement learning methods, which learn to choose actions to maximize their expected future reward, we apply simulation-based search algorithms to find the next move in an online search. These methods use a forward model to predict the outcome of an action. In case the forward model is not available it can be approximated during repeated interactions with the game. Such an approximated forward model allows us to apply methods such as Monte Carlo Tree Search (MCTS) [6] for finding actions that maximize the player's score or his chances in winning the game.

In a previous study we tested Forward Model Approximation in comparison with other agents on the General Video Game Artificial Intelligence-framework (GVGAI) [7]. It outperformed other agents, which were developed for the learning track of the corresponding competition, while being trained for significantly shorter training times. In our previous study the developed models were limited to a small subset of the state observations. Here, it proved to be beneficial to not just create a single model, but multiple models, each predicting different components of the game.

In this paper we expand on the concept of forward model approximation, by designing a model learning scheme in context of the GVGAI-framework. While the original model was limited to modelling only the player and its interactions, the new model architecture should be able to model all instances of an unknown game. This split into multiple models allows it to decrease the learning time, while improving the accuracy of the model.

The main contributions of the paper are:

- a theoretical justification of forward model approximation and its improvements based on model splitting
- explanations on how to decompose a model into multiple separate sub-models
- an analysis of the models accuracy
- an outlook on the agent model being developed using the approximated forward model

Section II briefly outlines the GVGAI framework and the many solutions that were submitted to the GVGAI competition. Section III reviews the basics of reinforcement learning and forward model approximation. Furthermore, we explain in detail how forward model approximation deviates from the classical reinforcement learning based approach. After that, in Section IV, we describe how model decomposition can be used to reduce the complexity of an approximated forward model. Results of the model decomposition procedure will be shown in Section V. In the same section we will evaluate the models accuracy in a set of games used in the GVGAI-competition. A conclusion and an outlook on future work are given in Section VI.

978-1-5386-9276-9/18/$31.00

II. General Game Playing

In contrast to learning an agent for a single game, general game learning focusses on the development of agents, which are able to learn and play a diverse set of games. Many early works studied games based on the Stanford General Game Playing competition [8], in which agents competed in various previously unknown games. Those were defined in a game description language and consisted of multiple logical rules for state-transitions, actions, and the number of players.

Due to the popularity and flexibility of video games, similar competitions were initialized for studying digital games. One of those is the GVGAI competition [7], which started in 2014. In the GVGAI competition games are defined using the Video Game Definition Language [9]. The competition framework currently provides about 100 arcade like video games. Agents can either read the graphical or logical output of the game and provide actions in the form of controller input.

In the game playing track agents are provided with a forward model of the game to be played. This allows the application of simulation-based search methods such as Monte Carlo Tree Search (MCTS), Rolling Horizon Evolutionary Algorithm (RHEA) [10], [11], and Open Loop Search (OLS) [12]. A complete overview of previously submitted agents can be found in [13]. Another part of the competition is the game learning track (since 2017), in which agents do not receive information on the forward model, but need to adapt their behaviour based on five minutes of interacting with the game. Here, decisions can only be based on the current state and the agent's knowledge of previous interactions with the game. Referring to the results of the competition in 2017, agents are either based on reinforcement learning techniques or use simple search schemes. Simulation-based search schemes cannot be applied without addressing the absence of the forward model.

In previous competitions the agents' performance in the game playing track is much better than in the game learning track. Agents are often able to win a game or at least find action sequences in the game playing track, which yield a high score, despite being confronted with a previously unknown game or level. Agents of the learning track often struggle with complex interactions between game entities or sparse rewards. While the learned models are sometimes performing well on the levels that were available during the training time, the performance often dropped when confronting the agent with previously unseen levels of already known games.[1]

In a recent paper we proposed a new learning scheme called forward model approximation [1], which tries to bridge the performance gap between agents of the playing and the learning track. During repeated interactions the agent learns an approximated forward model, which allows us to use simulation-based search schemes, as they were used in the game playing track. An approximated forward model with high prediction accuracy will allow us to find action sequences, which are likely to perform well in the real game.

[1]The analysis of learning track submissions is based on results presented on the competition website [20], because of a lack of accompanying publications.

III. Preliminaries

A. Action Environment Interface

The goal of reinforcement learning is to learn from repeated interactions, such that the chosen actions focus on achieving a pre-defined goal or maximizes an associated reward. In general the action-environment interface can be used to describe interactions between an agent and its environment. It consists of:

- **Agent:** the learner and decision-maker
- **Environment:** anything the agent can interact with. The environment can be modelled as a set of states $\mathcal{S}$, which change based on the chosen actions of the agent.
- **Actions:** the set of actions $\mathcal{A}$ for influencing or interacting with the environment.
- **Reward:** a numerical reward $r \in \mathbb{R}$ (provided by the environment), which is to be maximized

At each time-step t the agent is asked to choose an action $a_t \in \mathcal{A}$ based on the current state of the environment $s_t \in \mathcal{S}$, such that he maximizes the sum of received rewards over time. Further, the environment transitions into a new state s_{t+1} and returns a numerical reward r_{t+1} to the agent.

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

$$P(r_{t+1}, s_{t+1} | s_0, a_0, r_1, \ldots, s_t, a_t, r_t)$$

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 [14]. In other words the probability distribution for the successor state reduces to:

$$P(s_{t+1} | s_0, a_0, r_1, \ldots, s_t, a_t, r_t) = P(s_{t+1} | s_t, a_t)$$

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

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

Reinforcement learning methods try to estimate the expected reward for states or state-action pairs. The expected reward of a state action pair is the weighted mean of all possible state transitions and their associated reward:

$$\begin{aligned} r(s,a) &= \mathbb{E}[r_{t+1} | s_t, a_t] \\ &= \sum_{r \in \mathbb{R}} \sum_{s_{t+1} \in \mathbb{S}} R(s_t, a_t, s_{t+1}) \cdot R(s_{t+1} | s_t, a_t) \end{aligned}$$

The agent can now base its decisions on the maximal expected future reward or the sum of expected future rewards. Given enough time to learn an appropriate approximation of the expected reward reinforcement learning methods proved to effectively handle a wide range of decision-making tasks.

B. Forward Model Approximation

In contrast to reinforcement learning methods, forward model approximation does not try to approximate the expected reward but uses information on previous observations to increase its confidence in an approximated version of the forward model. In case the Markov property is fulfilled a forward model can be constructed by training a classifier to predict the future state based on the previous state and the chosen player action, therefore, estimating the probability distribution $P(s_{t+1}|s_t, a_t)$. The training of such a classifier is based on a database of previous interactions with the environment. In case the prediction accuracy is high enough, the learned model can be used to simulate future states and predict the outcome of an action sequence at run-time.

In general the complexity of learning such a classifier in an unknown game world is extremely huge. Most often the number of sensors, their possible values, and most of all the number of possible environment states is too huge to find an efficient mapping. Additionally, many environment states also mean that the number of training examples needs to be high, such that the classifier is able to generalize well. These criteria speak against the learning of such approximated forward models, but luckily environments in the context of games often impose restrictions that make this learning process much more feasible.

For example, games often display multiple entities that act mostly independent from each other (the player's agent, non-player characters, etc.). Predicting the next state for each of them at once using a single model would unnecessarily enlarge the classifier. Simpler models can be constructed by modelling every single entity individually and aggregating the results of all simple models to an overall state prediction later on.

A first study of the development of such a distributed approximated forward model is covered in [1]. The developed approximated forward model consisted of three sub-models. Those models represented the player's movement, the score change, and the winning/losing conditions of the game. An observation database was constructed by playing the game for approximately one minute using random actions. The generated model was stored as an ensemble of hierarchical knowledge bases [15], [16]. Finally, we used the approximated forward model in conjunction with Monte Carlo Tree Search to find profitable actions.

In our tests the agent outperformed previous submissions of the GVGAI-competition's game learning track by a large margin, despite being very restricted in its prediction capabilities. Games that included complex interactions between different game entities could not be simulated well, since the developed models did not consider changes in the environment other than the player's position. In games in which only short action sequences need to be planned, it was enough to consider other entities to be static. If they change their position, the next action plan can be adapted during the next turn. However, games in which long term planning was necessary it was still impossible to plan ahead due to the lack of prediction accuracy.

IV. Agent Model

In this paper we advance on our previous ideas and develop a scalable forward model architecture in the context of the GVGAI competition. Before we present our developed model we want to quickly review the model requirements:

- **Learning Speed:** For the application to the GVGAI-framework the models learning time is limited to a total of 5 minutes. This includes data collection and data processing. Previous submission showed that reinforcement learning approaches tend to learn too slow. We hope that classifiers that can learn a reliable prediction from a small set of examples may yield better agents.
- **Processing Speed:** Monte Carlo Tree Search and other simulation-based search algorithms benefit from many simulations. For this reason, the model needs to be fast in its prediction to allow multiple simulations in the short action time frame of 40ms. This time limit is strictly enforced by the GVGAI-framework for which we develop our agent.
- **Generalization:** It should be possible to use the model for the prediction of unseen situations or levels. Overfitting and underfitting of the learned model would hinder the generalizability, therefore, the developed model would be unsuitable to act as an approximated forward model.
- **Accuracy:** Utilizing the model as an approximated forward model used for simulation-based search methods requires a high prediction accuracy. Ensuring correctness of simulated episodes increases the certainty of the prediction and is a necessary requirement. Due to some games being probabilistic it will not always be possible to find a suitable model based on the recorded data, but iteratively improving the model should be reliable for all deterministic processes.
- **Interpretability:** While not being a strong requirement it would be beneficial to have an interpretable model. This would help the developers to detect apparent errors. It can also be a basis for human-computer interaction in which both a human player and the computer agent teach each other, e.g. teaching a new player the rules of an unknown game, pointing out mistakes in the prediction and give counterexamples.

Based on the requirements we decided to use the J48 classifier, which is available in the machine-learning software package weka [17], [18]. This classifier is an optimized implementation of the C4.5 decision tree [19] to generate pruned and unpruned decision trees.

A decision tree is a tree-based classifier and a popular choice in machine learning applications, because of it simplicity and speed. The inner nodes represent attribute conditions, whereas leave nodes represent the prediction of the classifier. Edges are associated with a decision made in an inner node. Decision trees can also be represented as a set of rules, whereas each path from the root node to any leave node represents a rule.

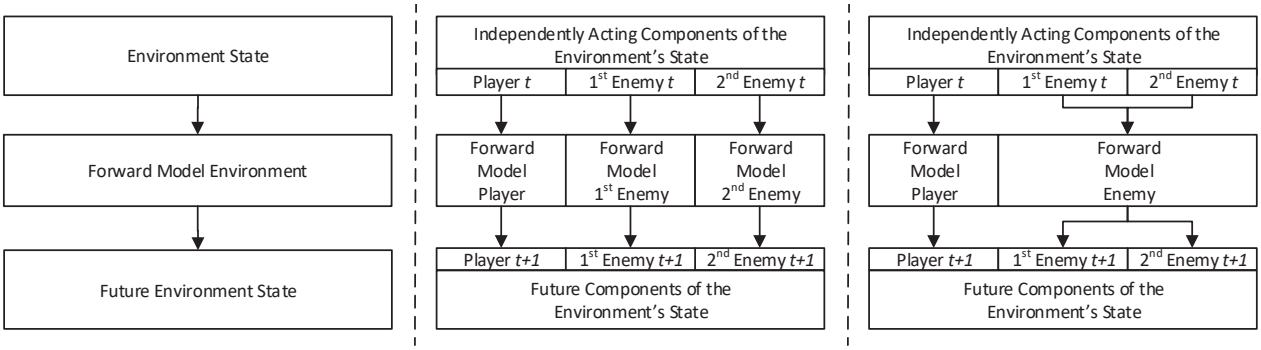

Fig. 1. Comparison of forward model architectures: (left) single complex forward model for the whole environment; (middle) forward model-ensemble consisting of a sub-model for each entity of the environment; (right) forward model-ensemble consisting of a sub-model for each type of entities

A. Model Architecture

As discussed in the Section III-B the architecture of the forward model can have a drastic impact on the training time and accuracy of the model. A single complex model learning all the possible interactions of game entities might be too complex to be learned efficiently. Instead of learning a single model we can reduce the problem complexity by learning a model for each independent component of a game. Those components can for example be defined by each actor/character in the game.

The idea behind this will be shortly explained based on the game alien, which is one of the games provided in the GVGAI-framework. Figure 2 shows the visualization of an aliens level. Here, the player controls a little spaceship, which can fly left and right or shoot a bullet upwards. Incoming alien spaceships need to be destroyed to win the game. Those traverse the screen from left to right or inverse. When they reached the end of their row they get down on the next row. In case the player fails to shoot all alien spaceships till at least one of them reaches the bottom of the screen he loses the game. The player scores two points in case he shoots an alien and one point when he shoots a boulder.

During the game alien spaceships will cover multiple positions on the screen. In general each of those position combinations forms a separate game state. However, predicting the next position of a single alien spaceship is completely independent of all other spaceships.

Fig. 2. Gamestate of the game "aliens" from the GVGAI-framework. Four types of entities: top: alien spaceships, middle: player shot and boulders, bottom: player spaceship

In case two entities act independent from each other it is simpler to model each entity on its own than modelling the combined movement of all entities. For this reason we refrain from learning a single complex model, but learn multiple simple models for predicting the next game state. Such a collection of single entity models first predicts the future state of each entity and then aggregates their results to create the fully predicted game state.

We can further optimize this procedure by grouping similarly behaving entities into a unified model. Instead of building a single model for each alien spaceship, all alien spaceships can share a common model (considering that they exhibit a similar behaviour). This not only decreases the number of models to be trained, but also increases the amount of available data for the model construction by including the data of all similar entities in a single training set. Figure 1 highlights the differences of the discussed forward model types.

Our final model for games of the GVGAI-framework was constructed using the following steps:

1) First we collect data for each entity by observing the game state using an agent, which is interacting randomly with the game. All observable entities in the game environment were recorded during a single playthrough of the three available training levels. Changes of an entity's attributes were stored in a database, which was later used for building a classifier.
2) Entities were grouped based on their typeID (provided by the GVGAI framework). The typeID represents a natural grouping of the Video Game Definition Language.
3) For each group we learn an independent forward model, which predicts the state transition of a single entity, based on the observed state transitions of all entities of the same type. Additionally, we learn one model to predict any score changes and another one to find the winning or losing conditions of the game.
4) For the prediction of the next gamestate we predict the state transition of each entity using its associated model. The results are aggregated by incorporating the state transitions of each entity into the current state.

B. Model Learning

As explained in the previous section we collect a dataset of entity observations for each type of entities. During the game we track changes for each entity and insert those into their type related data collection. In case a new object appears it is added to the data collection process of its specific type. An object is regarded to have been updated in case one of its target attributes changes (position change or object destruction). The time since the last update is tracked for each entity individually to model time-dependent state transitions.

During the data collection we store the position (pixel and grid position) of an entity, its neighbouring entities in the grid, and its relative position to the player. Additionally, we keep track of the object's last movement and the delay of its last update. In case an entity does not appear in one of the following game-states, it is considered to have been destroyed and, therefore, removed from the data collection process. For each entity type we store the number of created and destroyed instances per game tick. This information is later used to learn a separate model for predicting the score changes and analysing termination conditions. Table I and Table II list all variables that are tracked for non-player and player entities.

For each object type we learn three decision tree classifiers, two predicting changes in the x- and y-axis, and a third model to predict if an object will be destroyed in the next time-step.

Next to the prediction of each entity we also build a classifier to predict score changes and termination conditions of the game to be learned. Both are especially relevant for rating the outcome of an episode in a simulation-based search algorithm. Without a suitable scoring function the applied search scheme will not be able to differentiate winning from not-winning gamestates. The scoring function in the GVGAI-framework is fixed to interactions between various entities. While the framework does not provide any information on collisions between two entities, we can still try to measure the outcome of such collisions. Here, one or both of those entities are often removed from the game-state and the score is changed as a result, e.g. a bullet hitting an alien removes both entities and scores two points.

For this reason we are tracking all entity creations and destructions per type. The dataset-model for the general state classifiers is summarized in Table III. For predicting the score change we tested two model types. While a linear regression model seems to be an obvious choice for such a regression task, we observed that multiple scoring events occur only rarely. Hence, we also tested decision trees for predicting nominalized score changes as a second option. We did not observe a measurable performance difference. Because the decision trees were easier to interpret we restrict the remaining evaluation to the application of this classifier.

Predicting winning or losing conditions was based on the same dataset. However, the random bot only rarely wins a game, which would be necessary to find the win-conditions. Also, because the frequency of termination time-steps to continuouing time-steps is comparatively low ($\approx$ 1:1000) it is very unlikely that a pruned decision tree will incorporate necessary leaf nodes. This could be achieved by conditionalizing the dataset of state-transitions for the last time-step per game to detect rules for the termination of the game. Those rules need to be filtered later using a dataset of all remaining state-transitions. Only rules that are valid in the conditionalized dataset, but not in the remaining dataset can be considered to be termination rules. Nevertheless, to create a unified model architecture we refrain from applying a similar scheme here and base all models on a set of decision trees.

TABLE I
TRACKED NON-PLAYER ENTITY DATA

Instance Focused Attributes	Data Type
position	$\mathbb{N}^2$
grid-position	$\mathbb{N}^2$
reference vector	$\mathbb{N}^2$
last position change	$\mathbb{N}^2$
left-/right-/up-/bottom-neighbour type	$\mathbb{N}$
time since last update	$\mathbb{N}$

Player Focused Attributes	Data type
last player action	{left, right, up, down, use}
player distance	$\mathbb{N}^2$
player reference vector	$\mathbb{N}^2$

Target Attributes	Data type
changed position	$\mathbb{N}^2$
was object destroyed	{True, False}

TABLE II
TRACKED PLAYER DATA

Player Focused Attributes	Data Type
position	$\mathbb{N}^2$
grid-position	$\mathbb{N}^2$
reference vector	$\mathbb{N}^2$
last position change	$\mathbb{N}^2$
left-/right-/up-/bottom-neighbour type	$\mathbb{N}$
time since last update	$\mathbb{N}$
player action	{left, right, up, down, use}

Target Attributes	Data type
changed position	$\mathbb{N}^2$
was object destroyed	{True, False}

TABLE III
GENERAL STATE DATA

Player Focused Attributes	Data Type
created objects per type	$\mathbb{N}$ per type
destroyed objects per type	$\mathbb{N}$ per type

Target Attributes	Data type
changed score	$\mathbb{N}^2$
has game ended	{Player Won, Player Lost, Game Continues}

TABLE IV
PREDICTION ACCURACY ON THE GVGAI-LEARNING TRACK GAMES,
TRAINING LEVELS: ACCURACY OF ALL INSTANCES; VALIDATION LEVELS: ACCURACY OF ALL MOVING INSTANCES.

G1 = ALIENS, G2 = BOULDERDASH, G3 = BUTTERFLIES, G4 = CHASE, G5 = FROGS,
G6 = MISSILE COMMAND, G7 = PORTALS, G8 = SOKOBAN, G9 = SURVIVE ZOMBIES, G10 = ZELDA

	Mean Accuracy (unpruned / pruned)				
	G1	G2	G3	G4	G5
1x training level 2	**0.970 / 0.970**	**0.998 / 0.998**	**0.980** / 0.962	0.996 / **0.997**	0.466 / **0.992**
1x training level 3	0.991 / **0.998**	**0.999** / 0.998	0.933 / **0.947**	**0.997** / 0.995	**0.975 / 0.975**
1x validation level 1	**0.806** / 0.753	**0.636** / 0.616	**0.664** / 0.663	0.667 / **0.704**	0.700 / **0.803**
3x validation level 1	**0.773** / 0.663	0.616 / **0.621**	0.666 / **0.718**	**0.850** / 0.776	0.736 / **0.804**
5x validation level 1	**0.717** / 0.677	**0.593** / 0.583	**0.665 / 0.665**	**0.811** / 0.801	0.670 / **0.804**
1x validation level 2	0.707 / **0.708**	0.595 / **0.561**	**0.666 / 0.666**	0.645 / **0.667**	**0.667 / 0.667**
3x validation level 2	0.730 / **0.742**	**0.591** / 0.508	**0.666** / 0.665	0.671 / **0.692**	**0.667 / 0.667**
5x validation level 2	0.647 / **0.780**	**0.614** / 0.581	**0.744** / 0.723	0.667 / **0.671**	**0.667 / 0.667**

	Mean Accuracy (unpruned / pruned)				
	G6	G7	G8	G9	G10
1x training level 2	0.990 / **0.991**	**0.998 / 0.998**	0.994 / **1.000**	**0.996** / 0.995	**0.987 / 0.987**
1x training level 3	**0.996 / 0.996**	**0.998** / 0.997	**1.000 / 1.000**	**0.995 / 0.995**	**0.998** / 0.997
1x validation level 1	0.635 / **0.643**	**0.980** / 0.936	**0.667 / 0.667**	**0.670** / 0.604	**0.688** / 0.674
3x validation level 1	**0.654** / 0.647	**0.960** / 0.786	0.667 / **0.727**	0.648 / **0.687**	**0.783** / 0.721
5x validation level 1	0.640 / **0.667**	0.667 / **0.950**	**0.941** / 0.733	**0.714** / 0.679	**0.752** / 0.726
1x validation level 2	0.420 / **0.539**	0.773 / **0.798**	**0.667** / 0.633	0.692 / **0.707**	0.683 / **0.724**
3x validation level 2	**0.903** / 0.658	**0.837** / 0.769	**0.667** / 0.542	**0.749** / 0.721	**0.773** / 0.733
5x validation level 2	0.664 / **0.659**	**0.667 / 0.667**	0.667 / **0.733**	**0.699** / 0.646	0.735 / **0.751**

V. EVALUATION

We evaluated our model architecture on a set of ten games used in the GVGAI competition's learning track. In 2017 the games Aliens, Boulderdash, Butterflies, Chase, Frogs, Missile Command, Portals, Sokoban, Survive Zombies, and Zelda were used to compare the submitted agent's performance (see [20] for details). For each of these games the framework provides three levels for training and two levels for validation.

In this study, we did not focus on an agent's playing performance, but on its accuracy in predicting the next state. The accuracy measures the percentage of correctly predicted instances over the number of all instances to be predicted. Following the procedure proposed in Section IV we learn a model after every finished level and test its performance on the next level. This is done by recording all entities during the first 100 game ticks of the level (or less when the game ends early) and compare the model's prediction with the recorded future state of every entity per game tick. For each game we first play the three training levels only a single time. Afterwards both validation levels are played alternately for five times each.

After finishing the first training level we use the collected data to train a model for predicting every entity of the second level. In Table IV we compare the model accuracy of unpruned and pruned decision trees. Higher accuracy per game is highlighted in bold. After the second level a new model is built and used to predict every entity in the third level. During the training levels the accuracy for predicting all entities is nearly always ≥ 0.90, an exception being the game Frogs, in which the agent lost the first level after just a few ticks. This prevented him from collecting enough data for building a reliable model. However, the high accuracy values are not surprising, because the data set includes many static entities, which never move at all. A simple model that always predicts no state change would also score very high in this scenario.

To get a better insight on the capabilities of the proposed model architecture, we use the unseen levels of the validation set for further evaluation. Here, we do not test the prediction of all entities, but only on the entities that are known to change during the next state transition. Therefore, the prediction is only tested on entities that are about to change at least one of their target attributes. All these entities would be predicted incorrectly using the naive assumption of all entities being static, yielding an accuracy of 0. However, the trained model still predict changes of moving entities with high accuracy. Table IV lists the accuracy values of the first, third, and fifth iteration of playing each validation level.

In our tests there was only rarely a significant difference between a model of pruned and a model of unpruned decision trees. While pruning often helps to decrease the influence of noise in the classification, it can be beneficial to use an unpruned tree in deterministic games, because they can fully represent past training examples. Since, we do not know if the game being tested is deterministic or not, we cannot predict

which model should be used.

VI. Conclusion and Future Work

In this work we proposed a model architecture for forward model approximation in unknown games. The developed model architecture focuses on the GVGAI-framework, but can be applied to other reinforcement learning tasks as well. In contrast to creating a single model we propose to use a model ensemble. The architecture focuses on reducing the learning and prediction time by splitting the state into multiple independent components, which are each modelled by a simpler model. Those specialized models theoretically decrease the necessary training data and are faster to process.

The proposed model architecture was evaluated on ten games of the GVGAI-framework. The prediction accuracy of the trained models was generally high. Based on this result we plan to develop a simulation-based search on top of the trained model architecture. We hope that the prediction of all entities, instead of just the player, as done in [1], will further improve the performance of our developed agent.

While the proposed model performs very well in the context of the GVGAI framework it heavily relies on the character type information provided by the Video Game Definition Language. We plan to further extend the decomposition by studying dependencies between characters and their property values. An automatic detection of similar behaving components may further reduce the model size. It may also help to detect global effects or interactions between different game components, which are currently not modelled yet, e.g. a switch that opens a far away door.

Another interesting task it the development of multi-agent models. In scenarios were multiple players interact with each other it will be crucial to study the opponent player's behaviour to adapt the own behaviour. Such dynamic environments, which can be strongly related to game theory, pose additional requirements to the proposed model.

In the future we also plan to analyse the capabilities of different classifiers in the distributed model architecture that we used here. Especially the application of probabilistic classifiers would be interesting, which can detect if entities act randomly or provide confidence values for its prediction of future states.

References

[1] A. Dockhorn and D. Apeldoorn, "Forward Model Approximation for General Video Game Learning," in *2018 IEEE Conference on Computational Intelligence and Games, CIG 2018*, 2018.

[2] "OpenAI Five Blog Post," https://blog.openai.com/openai-five/, accessed: 2018-07-18.

[3] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez, M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. P. Lillicrap, K. Simonyan, and D. Hassabis, "Mastering chess and shogi by self-play with a general reinforcement learning algorithm," *CoRR*, vol. abs/1712.01815, 2017. [Online]. Available: http://arxiv.org/abs/1712.01815

[4] 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. [Online]. Available: http://dx.doi.org/10.1038/nature14236 http://www.nature.com/articles/nature14236

[5] 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*, 2018, pp. 621–632. [Online]. Available: http://link.springer.com/10.1007/978-3-319-91476-3_51

[6] 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. [Online]. Available: http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=6145622 http://ieeexplore.ieee.org/document/6145622/

[7] 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.

[8] 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.

[9] T. Schaul, "A video game description language for model-based or interactive learning," in *2013 IEEE Conference on Computational Inteligence in Games (CIG)*. IEEE, aug 2013, pp. 1–8. [Online]. Available: http://ieeexplore.ieee.org/document/6633610/

[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," 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. [Online]. Available: http://link.springer.com/10.1007/978-3-319-55849-3_28

[11] 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 2017*, 2017, pp. 88–95.

[12] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open Loop Search for General Video Game Playing," in *Proceedings of the 2015 on Genetic and Evolutionary Computation Conference - GECCO '15*. New York, New York, USA: ACM Press, 2015, pp. 337–344. [Online]. Available: http://dl.acm.org/citation.cfm?doid=2739480.2754811

[13] D. Pérez-Liébana, J. Liu, A. Khalifa, R. D. Gaina, J. Togelius, and S. M. Lucas, "General video game AI: a multi-track framework for evaluating agents, games and content generation algorithms," *CoRR*, vol. abs/1802.10363, 2018. [Online]. Available: http://arxiv.org/abs/1802.10363

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

[15] 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. [Online]. Available: https://aaai.org/ocs/index.php/FLAIRS/FLAIRS17/paper/view/15510/15038

[16] ——, "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.

[17] M. A. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA data mining software: an update," *SIGKDD Explorations*, vol. 11, no. 1, pp. 10–18, 2009. [Online]. Available: http://portal.acm.org/citation.cfm?id=1656274.1656278

[18] E. Frank, M. A. Hall, and I. H. Witten, "The WEKA Workbench," *Morgan Kaufmann, Fourth Edition*, pp. 553–571, 2016. [Online]. Available: https://www.cs.waikato.ac.nz/ml/weka/Witten_et_al_2016_appendix.pdf

[19] R. Kohavi and J. R. Quinlan, "Data mining tasks and methods: Classification: decision-tree discovery," in *Handbook of data mining and knowledge discovery*. Oxford University Press, Inc., 2002, pp. 267–276.

[20] "General Video Game Artificial Intelligence Competition," http://gvgai.net/, accessed: 2018-07-18.