Local Forward Model Learning for GVGAI Games

Alexander Dockhorn and Simon Lucas
School of Electrical Engineering and Computer Engineering
Queen Mary University of London
London, UK
a.dockhorn@qmul.ac.uk, simon.lucas@qmul.ac.uk

***Abstract*—In this paper, we are going to explain the design process for our GVGAI game-learning agent, which is going to be submitted to the GVGAI competition's learning track 2020. The agent relies on a local forward modeling approach, which uses predictions of future game-states to allow the application of simulation-based search algorithms. We first explain our process in identifying repeating tiles throughout a pixel-based state observation. Using the tile information, a local forward model is trained to predict the future state of each tile based on its current state and its surrounding tiles. We accompany this approach with a simple reward model, which determines the expected reward of a predicted state transition. The proposed approach has been tested using multiple games of the GVGAI framework. Results show that the approach seems to be especially feasible for learning how to play deterministic games. Except for one non-deterministic game, the agent performance is very similar to agents using the true forward model. Nevertheless, the prediction accuracy needs to be further improved to facilitate a better game-playing performance.**

***Index Terms*—Local Forward Model, Rolling Horizon Evolutionary Algorithm, General Game Learning, GVGAI framework**

I. Introduction

The General Video Game AI (GVGAI) framework [1] and its accompanying competition were created to study the development of general game-playing agents. Over the course of seven years, many competition tracks have been evolved, each focusing on a specific aspect of general game AI. The GVGAI competition's game-learning track, which was introduced in 2017, challenges agents to learn how to play several games by playing them. During training, an agent receives a visual representation of the current state and needs to choose its actions accordingly. It receives feedback in the form of rewards, and information on whether the game has been won or lost.

In contrast to the GVGAI competition's game-playing track, agents do not receive access to the game's forward model. Therefore, they are not inherently able to predict the outcome of their actions. This restriction has rendered agents, which rely on simulation-based search, ineffective. During the first two years of the learning-track, an agent's training process was limited to 5 minutes of real-time game-playing. None of the submitted approaches had resulted in significantly better game-play than a random agent [2].

These tough limitations on the agents' training time were lifted in 2019. Since then, agents can use unlimited training time given two training levels per game. This allowed the application of deep reinforcement learning agents, which have been shown capable of learning to play several games of the GVGAI framework [3]. Deep reinforcement learning agents train a neural network to approximate the expected return of an action. Due to a large number of model parameters, these techniques often require a lot of training time and a diverse set of training examples. While [3] has shown successful applications for some GVGAI games, results of reinforcement learning agents are often overshadowed by the agents' performance in the competition's game-playing track [2].

In this paper, we are going to explore the applicability of forward model learning to apply simulation-based search in a game-learning scenario. With forward model learning the agent tries to predict the outcome of its actions based on previously observed interactions with the environment. During training, the outcome of each action is observed and used to approximate the environment's response in terms of state and reward changes. In contrast to reinforcement learning-based approaches, the agent determines the value of its actions at run-time by rating action sequences based on their predicted outcome. For this purpose, any search algorithm can be used in conjunction with the trained forward model simulating the action's influence on the environment.

In recent works, many attempts have been made to learn accurate and reliable forward models for various game-like environments. Ha and Schmidhuber [4] have used auto-encoders and recurrent neural networks to predict the upcoming frames of several games of OpenAI's gym framework [5]. Similarly, generative state-space models [6]–[9] have been used to facilitate planning in simulated environments. Instead of modeling the outcome of a state transition, neural GPU models [10] have been used to model the operations required for each state transition. These models have proven to create reliable models for multiple games of the GVGAI framework [11].

Last on our list are local model approaches. Those consider a game-state transition to consist of many independent local interactions among observable entities. While the independence assumption limits the generality of local forward models, it also results in considerably higher sampling efficiency. This makes them especially interesting for the application in the GVGAI learning track since the number of available training levels is very limited. In [12] and [13] a local forward model has been used to model state transitions of the Game of Life and Sokoban, respectively. Initial tests on games of the GVGAI

978-1-7281-4533-4/20/$31.00 ©2020 IEEE

(a) golddigger (b) treasurekeeper (c) waterpuzzle

Fig. 1: initial game-state of the first level of each game of the 2020 GVGAI's game-learning track

framework [12] have shown that local forward models quickly identify a large number of predictable patterns, but require more work in terms of generalization to unknown game-states.

In this paper, we are further going to explore the capabilities of local forward agents and how they can be trained efficiently. The main contributions of this paper are:

- an automatic pre-processing of pixel-based state observations into tile-based state observations
- an efficient process for learning local forward models for deterministic games
- a simple reward model based on predicted state transitions
- a case-study on learning local forward models in the context of multiple games of the GVGAI framework

To better explain the context of this work, we will first describe the GVGAI framework in Section II and the games of the 2020 GVGAI competition's learning track in Section II-A. The remainder of this paper will focus on the design process of our proposed agent (Section III), its training (Section IV), and the evaluation of its game-playing performance (Section V) followed by our conclusion in Section VI.

II. GVGAI FRAMEWORK

The General Video Game AI (GVGAI) framework [1], developed in 2014, was created to provide a unique test-bed for general game-playing agents. Games are defined using the Video Game Definition Language [14], which allows the description of 2-dimensional arcade-like video games. At the time of writing this work, the GVGAI framework provides a unified interface for more than 100 games.

Experiments of this paper will be based on the Python client of the GVGAI single-player learning track [15]. While games are run on a Java server, the client receives a visual representation of the current state, the set of available actions, the agent's reward, and, in case of a terminal state, additional information on the game's outcome (win, loss, or timeout).

A. GVGAI Learning Track

The 2020's GVGAI learning track features three games, namely *golddigger*, *treasurekeeper*, and *waterpuzzle*. In the following, we will shortly introduce these games.

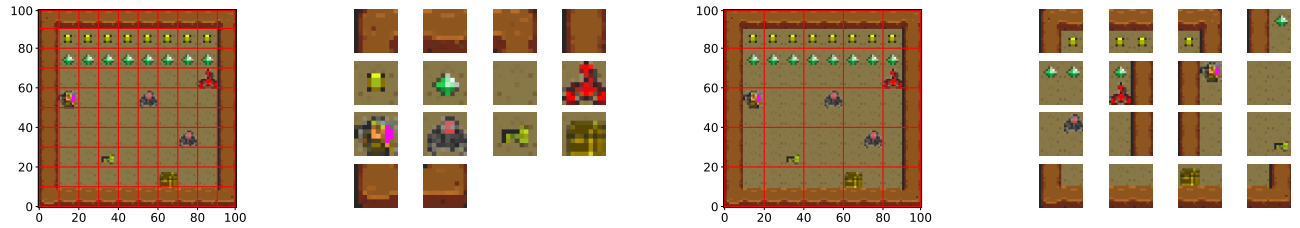
In *golddigger* (Fig. 1a) the agent needs to dig out gold coins and diamonds while staying away from multiple randomly moving enemies. In contrast to many other games, the agent's avatar can only move in its current viewing direction. In case the agent attempts to move in another direction, it first changes its orientation (indicated by a small arrowhead). Using the action a second time moves the character in the indicated direction. After collecting a key, the agent can open chests to increase its score. However, some of the chests are traps and will transform into an enemy when the agent attempts to open them. Additionally, the game includes dirt tiles, which block the agent's movement. Dirt tiles can be removed by the agent using a shovel, which will destroy the neighboring dirt tile in direction of view. Similarly, the action can be used to gather neighboring gold coins and diamonds or to kill enemies for scoring points. The agent wins the game in case all coins, diamonds, and chests have been collected and loses the game when getting in contact with an enemy.

The game *treasurekeeper* (Fig. 1b) turns the concept of the previous game up-side-down. Here, the randomly moving enemies try to steal the treasure chest from the player. Since the agent has no direct way of stopping the enemies, it needs to push boxes around to block their path, while trying to avoid touching enemies. Every 100 game-ticks the agent receives 5 points and wins the games after surviving 600 game-ticks with at least one treasure chest remaining.

In *waterpuzzle* (Fig. 1c) the agent needs to traverse a maze to reach the exit. The latter is marked by a closed door, which can only be opened using a key. Collecting the key rewards the agent with 5 points, while opening the door rewards an additional 10 points and wins the game. The game is lost after failing to escape the maze in less than 1500 time-steps.

For each of these games, the competition provides two training levels, of which the first level is shown in Fig. 1. Before the final competition, the submission page ranks agents according to their performance on two previously unseen test levels. This allows tuning the agent's parameters without overfitting it to the secret test levels. The final ranking will be determined using a third validation level. While being unavailable at the time of writing this paper, these levels will be made available as soon as the competition finishes.

The agents' ranking is determined by their win-rate, their average score, and the length of played games. The highest priority is given to maximizing the agent's win-rate. If two agents perform similarly well, the second criterion is the number of points they have achieved (max). The third tie-



(a) slicing using tile size=10 (b) tile map for tile size=10 (c) slicing using tile size=20 (d) tile map for tile size=20

Fig. 2: Extracting tile maps for a simplified game-state of the GVGAI game *golddigger* for various tile sizes.

breaker is the agent's average number of ticks until a game has been won (min), while the last distinguishing feature is the average number of ticks until a game has been lost (max).

III. Agent Design Process

The proposed agent model relies on the accurate prediction of future game-states to enable the application of simulation-based search. Search algorithms have shown good results in many of the GVGAI competition game-playing in which the game's forward model is accessible to the agent.

In this section, we describe the underlying design process for our agent model. Specifically, we will first introduce the preprocessing of the state observation in Section III-A. Given the preprocessed state, we describe how the agent will use local forward models to predict an action's outcome (Section III-B). Using a reward model, the agent will be able to predict the expected reward of a state transition given the changes in between the original state and the predicted upcoming state (Section III-C). After the detailed introduction of each component, we will introduce the agent model in Section III-D and our devised training procedure in Section IV.

A. Preprocessing the State Observation

During each time-step of the game, the agent is being provided with a pixel-based representation of the current game-state. The provided image represents the whole level, such that no camera movement needs to be considered throughout playing the game. While a local forward model could directly be applied to the pixel information, the state of GVGAI games can often be condensed to a more efficient representation, i.e. a tile-based representation.

Many 2D-games use tile maps to represent levels. A tile set is a collection of tiles, each representing an elementary sub-graphic of usually equal size. In a tile map, each of the unique tiles is referred to by a unique ID. Instead of defining, the level in terms of a pixel-based representation, it is composed of each tile's ID. In fact the GVGAI framework implements the video game definition language (VGDL), which in turn uses a tile-based representation to encode levels.

Since the agent is unaware of the underlying tile map it will be required to reconstruct it throughout the course of its training. In case the agent knows the tile size, this process is fairly simple. The provided image can first be cut into evenly sized tiles. Furthermore, a tile map is created by collecting all unique tiles among these cuts and assigning them an ID. Since the tile size is unknown to the agent, we repeat this process for all true divisors of the input image's width and height. In Fig. 2 we show this process and its result given an exemplary game-state of the game *golddigger* for two different tile sizes. The process will be repeated for every observed game-state to construct a more reliable tile map.

Extracted tile maps are compared given their number of unique tiles and their tile size. While a small number of unique tiles helps to keep the final model simple, a large tile size is desirable to reduce the size of the input matrix as much as possible. For all the provided games, a tile size of 10 has shown to be a good trade-off between tile size, number of unique tiles, and the resulting model's complexity. In fact it resembles the framework's original tile size. The extracted tile map is further used to reduce the pixel-based observation[1] of size $(n, m, 4)$ to a tile-based observation of size $(n/10, m/10)$.

1) Noisy Images due to Image Compression: During extensive testing of the approach described above, we saw that some of the extracted tiles are very similar to each other. This can be the result of an object's animation-steps or noise introduced due to image compression or anti-aliasing. To get a more stable result, we relaxed the equality-condition for extracted slices. Instead of requiring two tiles to be completely similar to each other, we consider two tiles to be the same in case the correlation between their pixel values if higher than a given threshold. Good results were achieved for setting the threshold at a value of 0.85, which was low enough to identify all semantically different tiles, but high enough to merge different representations of the same tile.

B. Local Forward Models for Predicting Upcoming States

Local forward models have first been studied in the context of Conway's Game of Life [12]. The Game of Life is a cellular automaton in which an initial configuration evolves given a simple set of rules. Hereby, every cell interacts with its eight neighbors (Moore neighborhood) to determine its future state. In turn, the next game-state can be determined by separately handling each cell and aggregating the results. Both the locality

[1] for which the first two dimensions describe the position of a pixel and the third dimension its RGB-color and alpha-channel

(a) example state waterpuzzle

(b) extracted neighborhood patterns of the local forward model

(c) predicted result for action *left*

(d) score model input vector

occurrence count	tile type			
				
before	6	4	1	1
after	6	5	1	0
created	0	1	1	0
destroyed	0	0	1	1

Fig. 3: Predicting the next game-state using a cross-shaped neighborhood of span 1 and the reward model's input pattern.

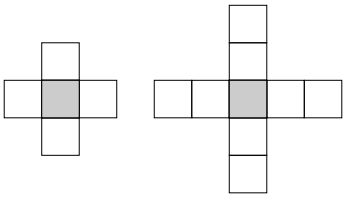


Fig. 4: Local neighborhood patterns of span 1 and 2 (including the centre tile) used to predict the next state of the centre tile.

of a cell's interaction and the aggregation process, are the underlying principles of local forward models.

A local forward model predicts the upcoming state of a cell given the state of its neighboring cells, whereas the neighborhood pattern can deviate from the classical Moore neighborhood. In this work, we will be using the cross-shaped pattern shown in Fig. 4, which has been successfully applied in a previous study on the game Sokoban [13]. The span length will be chosen according to the game being played.

1) Recording a Training Data Set: In the context of the proposed agent, we will be using the tile-based state representation constructed in Section III-A as an input of the local forward model's learning process. Given an observed state transition, we extract the neighborhood pattern of each cell and its resulting future state. Since the outcome can be dependent on the agent's action, we extend the neighborhood pattern by the action used during the last state-transition. The resulting input pattern and its recorded outcome are added to the models training data set.[2] The pattern extraction and prediction process is shown in Fig. 3a-c.

In the case of a non-deterministic game, the outcome of a pattern can vary in between multiple observed transitions. As a result, we will not store the latest observed outcome, but the occurrence count of all observed outcomes per input pattern.

2) Learning a Local Forward Model: Given the training data recorded in the previous subsection, we can render the prediction of upcoming tiles as a classification task. For a deterministic game, we learn a model that maps an input pattern, consisting of a cell's neighborhood and the agent's action, to the future state of the cell. We chose to use decision trees since they are quickly generated, can be applied fast and allow batch-processing of multiple input patterns during the prediction of upcoming states.

In the case of a non-deterministic game, we train a probabilistic classifier that maps the input pattern to the probability of the middle cell to become a certain tile type. When asked to predict an upcoming game-state, we sample the outcome of each cell respective to their expected probability vector.

Since the prediction of each cell is independent of the future state of other cells, the probabilistic prediction can yield unintuitive results. Fig. 5 shows an exemplary game-state, which consists of a single randomly moving enemy. Since the enemy could move in every cardinal direction, the cells below, right, left, and above the enemy have the chance of containing the enemy during the next turn. The probabilistic nature of predicting the next turn can result in the enemy disappearing or duplicating, as seen in Fig. 5d. Therefore, the application of a repair operator may be necessary to assure the validity of a predicted game-state. However, generating or learning such repair operators is beyond the scope of this work.

C. Reward Model for Predicting Upcoming Rewards

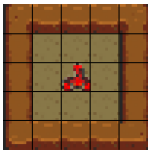
Next to the agent's local forward model, which predicts upcoming states, we implement a reward model to predict upcoming rewards of predicted state transitions. The input of our reward model will be the states before and after a state transition. In GVGAI games, tiles often represent separate game objects. Since reward functions of the GVGAI framework are often linked to interactions between such objects, we chose to use the occurrence counts of all tiles in the tile map to predict the reward.

For each tile in the tile map we extract the following values:

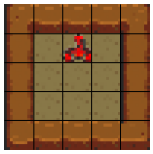
- its occurrence count in the state before the transition
- its occurrence count in the state after the transition
- the number of tiles that have become this tile type
- the number of tiles that are no longer of this tile type

Fig. 3d shows an exemplary game-state transition of the game *waterpuzzle* and the resulting occurrence counts per tile type. The values of the table are flattened into a one-dimensional

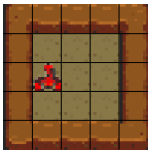
[2]Note, that the term *neighborhood pattern* refers to all cells considered to be neighbors of the current cell, while the term *input pattern* includes the agent's action.



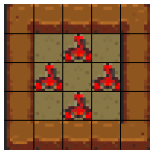
(a) initial state



(b) observed result



(c) valid prediction



(d) invalid prediction

Fig. 5: example states of the game *golddigger* and the probabilistic predictions of a local forward model

input vector for the reward model. Additionally, we record the observed reward received during the state-transition to mark the expected output of said input vector. Finally, any regression or classification model can be used to predict the reward based on the extracted occurrence vector.

To further highlight the importance of terminal states, we chose to modify the expected output of the reward model. For profitable game state transitions, we add the arbitrarily chosen large number 1000 to the expected output, while we subtract 1000 for transitions that result in a loss. We observed that this change helped the agent in prioritizing action sequences that let the agent win, while strictly avoiding situations in which it expects to lose.

D. Agent Model

In the following, we describe how the components introduced in the previous subsections are combined in our proposed agent.

Given a new game-state, the agent will first convert the pixel-based state observation into a tile-based state observation. During training, each tile that has not yet been observed by the agent is added to the agent's tile map and given a unique identifier. At the time of evaluating the agent, we will not add further tiles. Instead, we classify each tile according to its closest match with previously observed tiles. This results in problems in case new objects are introduced in later levels. However, adding new classes to the tile map would confuse the classifier when predicting the next state.

By replacing the game's true forward model with a combination of a trained local forward and reward model, we are able to predict the expected outcome of the agent's actions. Since no such model will be available right from the start of the training, the first interactions with the game-state can be done at random. As soon as the results of these actions have been observed, we can record observed patterns, compile a training data set, and train the respective models. Details on the agent's training process can be found in Section IV.

In case a model is already available, we can apply any search method to optimize the agent's action sequence. Given an action and a tile-based game-state, we first apply the local forward model to predict the expected upcoming game-state. Using both, the current and the upcoming game-state, as input for the reward model, we can predict the expected reward of said transition. Predicting the resulting score of a single action allows a greedy action-selection. However, since we predict the upcoming state, we can consecutively apply these models to predict the outcome of whole action sequences, and therefore, apply any search method. In this work, we will be using breadth-first search (BFS) for games that appear deterministic and the rolling horizon evolutionary algorithm (RHEA) for non-deterministic games. At the beginning of the agent's training each game is considered to be deterministic until any observation proves him wrong. Both search algorithms have previously been applied in agents of the GVGAI competition's game-playing track [2], [16], [17].

Note that, during the search we do not require further applications of the tile map since all computations are based on the tile-based state representation. Nevertheless, the tile map allows us to visualize any of the predicted states by replacing each cell with the pixel-based representation of its encoded tile. This allows us to visualize the agent's search path and retrace the choice of an action sequence based on its predicted outcome.

After the action has been applied to the game and the agent observes the true outcome of its action, the search tree can be updated in case of mistaken predictions. Furthermore, anytime a previously unknown pattern has been observed, we can update the models' training data given the new observation. Unfortunately, the tight time constraints of the competition (40ms per game-tick) do not allow us to update the model during evaluation. Hence, we will omit this step.

IV. Training the Agent

The framework provides two training levels for each of the competition's games. Since the GVGAI framework allows the generation of new levels, we create additional training levels by mirroring and rotating the original levels. This allows us to diversify the training data and teach the agent about symmetric relationships in its local forward and reward model. All the level files and a visualization of their initial game-state can be found in the project's GitHub repository [18].

We began the training by building a data set based on observed interactions of a random agent. However, this resulted in a poor exploration of levels and yielded models which were rarely able to predict previously unseen patterns correctly. For example, in the game *golddigger* the random agent never reached the gold coins, which were several tiles away from its starting position. While we could technically give the agent more training time and therefore increase the likelihood of observing rare patterns, we chose to improve the efficiency of the training process. We want to achieve this by giving the agent an internal drive to increase the number of unique observed patterns for the local forward and the reward model.

We achieved this, by constructing a search tree based on the agent's previously observed patterns. For each state in which

all patterns are known, we extend the search tree by another child-node. In case a state includes unknown patterns, we count the number of unique patterns and use this as the explorative value of that state. Each time a new pattern has been observed, we check for all the states that can be expanded and recursively add their child nodes until we reach an unexpandable node.

Especially in deterministic games, this search process yields a more diverse exploration of the game-tree. Additionally, the agent can decide to stop the level's exploration as soon as it is known that there are no more patterns to be explored.

In case of non-deterministic games, we face the problem that the agent can never be sure that another rare outcome for a given pattern exists. Nevertheless, adding more observations increases the agent's confidence in its estimation of pattern outcomes. Therefore, during training, we favor states including unknown patterns but rely on a random exploration in case all patterns are known.

V. Evaluation

Submitted agents will be validated based on their performance in two test levels. The final ranking will be determined based on the agent's performance in a third validation level. Since these are unavailable at the time of submitting this paper, we can only provide the agent's results in the two training levels. Additionally, we validate the proposed approach using six other games of the GVGAI framework. We specifically selected games that implement similar game-mechanics as the three test games of the competition's learning-track. In the following, we shortly introduce chosen games.

The games *labyrinth* and *labyrinthdual* were chosen due to their similarities with the game *waterpuzzle*. They represent games in which a maze needs to be traversed by the agent. In *labyrinth* the agent needs to reach a flagpole while avoiding to step on any of the spike-traps. In *labyrinthdual* the maze's paths can be blocked by colored obstacles. These can only be passed in case the agent wears a cloak of the same color. Similarly colored houses allow the agent to change its cloak color accordingly but disappear after they have been used.

The games *bait* and *sokoban* were selected since they require the agent to push objects to designated locations. The same skill is required in the game *treasurekeeper*. In *bait* the agent needs to collect a key to escape the dungeon by opening the door. To reach the key, the agent needs to push boxes out of the way. Later levels include holes that can be filled by pushing a box into them. Similarly, the game *sokoban* requires the agent to push boxes to a target tile. As soon as a box is pushed on a target, the box disappears and the agent receives a point.

The games *boulderdash* and *zelda* share similarities with the game *golddigger*. In *boulderdash* the agent can remove blocks by digging through them. Additionally, collecting diamonds rewards the agent with points, while touching an enemy loses the game. In *zelda* the agent needs to escape a dungeon through a door. Both games allow the agent to kill neighboring enemies to score more points. Since all enemies are moving in random directions, these games are non-deterministic.

For the deterministic games *waterpuzzle*, *labyrinth*, *labyrinth-dual*, and *bait* we applied the optimized search process described Section IV. The game *sokoban* uses a deterministic rule-set as well. However, its pixel-based state observation does not allow to discriminate the case in which the agent's avatar is standing on an otherwise empty floor tile or a target position. Therefore, the game looks non-deterministic to the observer. As a result, we use the training process for non-deterministic games on the games *sokoban*, *golddigger*, *treasurekeeper*, *boulderdash*, and *zelda*.

During the evaluation, we use a BFS with 100 expansions per game-tick for deterministic games and RHEA with a horizon of 5 and 20 candidate solutions per game-tick for non-deterministic games. This results in the same number of forward model calls per game-tick. However, the BFS agent is able to achieve a higher search depth, due to continuously expanding the search-tree over the course of multiple game-ticks. The relatively small number of expansions was chosen to complete the search during each tick in about 40ms. Both agents use a discount of 0.99.

A. Results

For the games *waterpuzzle*, *treasurekeeper*, and *golddigger* we report the agent's average score on 20 runs per level. The results are compared to results of agents trained with one of the reinforcement learning algorithms DQN [19], A2C [20], and PPO2 [21] as well as base-line implementations of Random Search (RS), RHEA, Monte Carlo Tree Search (MCTS) [22], and OpenLoop MCTS (OLETS) [23] using the true forward model. Their performance values, as well as the baseline performance of a random agent, have been taken from the competition framework's GitHub repository [24]. Table I shows the average score of all tested agents on the two levels they were trained on.

A similar evaluation was done for the six other games of the GVGAI framework. Here, we focused on the comparison with search-based algorithms to highlight the performance differences of replacing the true forward model with a learned approximation. In Table II, we recorded the agent's win-rate and the average score per agent over 20 runs per level. Note, that the proposed agent has only been trained on the first two levels (and its rotated and mirrored copies).

B. Discussion

Our results show that the proposed agent is capable of learning to play several games of the framework based on a pixel-based state observation. It performed best for deterministic games in which an accurate forward model has been extracted. This is especially true for the maze-like games *waterpuzzle*, *labyrinth*, and *labyrinthdual* in which the agent performed best among all tested agents. Here, the forward model consists of the avatar's movement and the collection of items. Such interactions in a tile-based state perfectly match the assumptions of the local forward model and therefore are easy to extract during training. In return, they transfer well to previously unobserved levels, which is indicated by the

TABLE I: Agents' average scores on games of the 2020 GVGAI competition's game-learning track. Results were computed based on 20 consecutive runs of each agent. Reported values of reinforcement learning agents, true forward model agents, and the random agent were taken from the competition's Github repository [24]. Values of the best agent(s) per row were highlighted for better visibility.

Game	Level	Proposed	Reinforcement Learning			True Forward Model				
		Agent	DQN	A2C	PPO2	RS	RHEA	MCTS	OLETS	Random
waterpuzzle	0	**15.0**	0	0.5	0	5.5	6.5	8.5	**15**	3.5
waterpuzzle	1	**15.0**	0	1	0	4.75	4.75	6.75	11	2.5
treasurekeeper	0	7.75	**30**	2.5	17	2.25	2.4	2.35	0.45	0.75
treasurekeeper	1	**6.0**	2.25	2	0.75	1.75	2	1	0	0.75
golddigger	0	7.45	-1.7	15.9	-2.9	149.6	130	154	**164.6**	4.8
golddigger	1	4.4	0	8.15	-3.25	64.9	61.5	67.4	**88.9**	8.2

TABLE II: Results (win-rate / average score) on six games of the GVGAI framework. Results were computed based on 20 consecutive runs of each agent. Values of the best agent(s) per row were highlighted for better visibility.

Game	Level	Proposed	True Forward Model				
		Agent	RS	RHEA	MCTS	OLETS	Random
labyrinth	0	**1.0 / 1.0**	0.05 / -0.8	0.0 / -0.9	0.0 / -0.95	0.05 / -0.85	0.0 / -0.85
labyrinth	1	**1.0 / 1.0**	0.0 / -0.45	0.0 / -0.55	0.0 / -0.55	0.0 / -0.8	0.0 / -0.65
labyrinth	2	**1.0 / 1.0**	0.0 / -0.85	0.0 / -0.85	0.0 / -0.9	0.0 / -0.85	0.0 / -0.9
labyrinth	3	**1.0 / 1.0**	0.2 / -0.55	0.1 / -0.7	0.1 / -0.7	0.15 / -0.65	0.1 / -0.6
labyrinth	4	**1.0 / 1.0**	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0
labyrinthdual	0	**1.0 / 7.0**	0.0 / -0.65	0.0 / -0.25	0.0 / -0.8	0.0 / -0.75	0.0 / -0.3
labyrinthdual	1	**1.0 / 4.0**	0.0 / 1.9	0.0 / 1.6	0.0 / 1.7	0.0 / 1.55	0.0 / 1.7
labyrinthdual	2	**1.0 / 7.0**	0.0 / 2.0	0.0 / 2.0	0.0 / 2.0	0.0 / 2.0	0.0 / 2.0
labyrinthdual	3	**0.75 / 5.75**	0.0 / -0.85	0.0 / -0.85	0.0 / -1.0	0.0 / -1.0	0.0 / -0.85
labyrinthdual	4	**1.0 / 4.0**	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0	0.0 / -1.0
bait	0	**1.0 / 5.0**	0.3 / 1.5	0.4 / 2.0	0.35 / 1.75	0.35 / 1.75	0.3 / 1.5
bait	1	**0.3 / 2.75**	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0	0.0 / 0.0
bait	2	**0.0 / 0.6**	0.0 / 0.4	0.0 / 0.35	0.0 / 0.45	0.0 / 0.2	0.0 / 0.35
bait	3	**0.0 / 3.25**	0.0 / 2.05	0.0 / 1.6	0.0 / 1.9	0.0 / 1.7	0.0 / 1.95
bait	4	**0.0 / 3.4**	0.0 / 2.45	0.0 / 2.3	0.0 / 2.3	0.0 / 2.15	0.0 / 1.9
sokoban	0	0.0 / 0.05	**0.0 / 0.15**	0.0 / 0.05	0.0 / 0.0	0.0 / 0.1	0.0 / 0.0
sokoban	1	0.0 / 0.2	**0.0 / 0.45**	0.0 / 0.15	0.0 / 0.2	0.0 / 0.3	0.0 / 0.15
sokoban	2	0.0 / 0.85	0.0 / 1.05	0.0 / 1.1	0.0 / 1.0	0.0 / 0.9	**0.0 / 1.2**
sokoban	3	0.0 / 0.25	0.0 / 0.5	0.0 / 0.5	**0.0 / 0.6**	0.0 / 0.4	0.0 / 0.2
sokoban	4	0.05 / 1.0	0.05 / 1.0	0.05 / 0.9	**0.2 / 1.15**	**0.2 / 1.15**	0.05 / 0.95
boulderdash	0	0.0 / 0.55	0.0 / 0.9	0.0 / 1.05	**0.0 / 1.2**	0.0 / 0.2	0.0 / -0.1
boulderdash	1	0.0 / 2.3	0.0 / 2.0	**0.0 / 3.1**	0.0 / 2.4	0.0 / 2.7	0.0 / 2.15
boulderdash	2	0.0 / 3.65	**0.0 / 4.2**	0.0 / 3.2	0.0 / 3.0	0.0 / 3.65	0.0 / 3.2
boulderdash	3	0.0 / 0.3	0.0 / 0.75	0.0 / 0.15	0.0 / 0.25	**0.0 / 1.25**	**0.0 / 1.25**
boulderdash	4	0.0 / 1.9	0.0 / 2.4	0.0 / 2.4	0.0 / 1.3	0.0 / 2.15	**0.0 / 2.5**
zelda	0	0.05 / -0.15	0.0 / -0.15	0.0 / -0.05	**0.0 / 0.0**	0.0 / -0.05	0.0 / -0.4
zelda	1	0.0 / -0.05	0.0 / 0.0	0.0 / 0.7	0.0 / -0.1	0.05 / 0.0	**0.0 / 0.85**
zelda	2	0.0 / -0.15	0.0 / -0.3	0.0 / 0.2	0.0 / -0.2	**0.0 / 0.4**	0.0 / 0.15
zelda	3	0.0 / -0.3	0.0 / -0.1	0.0 / -0.05	0.0 / -0.1	**0.0 / 0.45**	0.0 / -0.4
zelda	4	0.0 / 0.0	0.05 / 0.35	0.0 / -0.25	**0.0 / 0.45**	0.0 / -0.25	0.05 / 0.4

nearly perfect win-rate of the agent among all three games. Similarly, the proposed agent was able to play the game *bait* better than other agents. For level 0, the perfect win-rate can be explained by the small number of moves required to finish a level. Since the applied breadth-first search always detects a valid solution during the first turn, the agent is able to reliably complete the level. The higher win-rate in these deterministic games can probably be attributed to the agent's search process as well. Due to the deterministic outcome, the agent is able to expand the search tree with every tick and finally find a winning action sequence. Nevertheless, this would not have been possible in case the learned forward model represents the game's mechanics poorly. To the best of our knowledge, the sample implementations of the other search algorithms did not implement similar optimizations.

In the game's *sokoban* and *treasurekeeper* the agent extracted a reliable forward model but lacked the ability to model the games' termination conditions. Especially in the case of *treasurekeeper* the agent has shown to flee from enemies but failed in blocking the enemy's path to the treasure chest. Both games require planning over very long horizons to win the game, which the agent is not capable of due to the

limited search time. This problem is also reflected in the poor performance of other search-based agents, which fail to win the game despite having access to the true forward model.

Similar problems arise in the games *golddigger*, *boulderdash*, and *zelda*. As long as the the collectibles remain out of the agent's reach (in terms of planning horizon), it will move randomly. This changes abruptly when the agent gets near an item. Especially in the first level of *golddigger*, an agent can follow the line of gold coins and diamonds until all of them have been collected. Sadly, the learned reward model failed in predicting the benefit of collecting these items such that our proposed agent was limited to a random exploration.

VI. Conclusion

In this paper, we presented our forward model learning agent for the 2020 GVGAI competition's game-learning track. Given a tile-based state representation the agent has shown to be able to learn reliable forward models of several deterministic games using a local forward model learning approach. For non-deterministic games, the agent was less successful due to not being able to capture the random movement of enemies reasonably well.

In the future, we would like to expand on our evaluation by using the trained forward model on each of the included search-based algorithms. From this, we expect further insights into the effects of learning an imperfect approximation to the true forward model and the performance of the applied search process. Additionally, we would like to further expand on the various components introduced throughout this paper.

In Section III-A1 we have introduced a method for merging similar looking tiles into a semantically equal tile type based on a minimal correlation of their pixel values. We would like to further expand on this approach by letting the agent decide if two visually different tiles should be merged based on their similar semantics. For this purpose, the agent may extract all unique tiles and merge two tiles in case the resulting local forward model achieves a higher prediction accuracy (or the agent achieves a higher game-playing performance). Similarly, we would like to improve the efficiency of training the agent to play non-deterministic games as well as its final game-playing performance. Furthermore, we would like to explore new ways in representing the reward model, since the two-phase approach slows down its evaluation and fails to express some game-mechanics (e.g. termination conditions of *treasurekeeper*).

References

[1] D. Perez-Liebana, 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," feb 2018.

[2] D. Perez-Liebana, S. M. Lucas, R. D. Gaina, J. Togelius, A. Khalifa, and J. Liu, *General Video Game Artificial Intelligence*. Morgan & Claypool Publishers, 2019, vol. 3, no. 2, https://gaigresearch.github.io/gvgaibook/.

[3] R. R. Torrado, P. Bontrager, J. Togelius, J. Liu, and D. Perez-Liebana, "Deep reinforcement learning for general video game ai," in *Computational Intelligence and Games (CIG), 2018 IEEE Conference on*. IEEE, 2018.

[4] 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.

[5] G. Brockman, V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang, and W. Zaremba, "Openai gym," 2016.

[6] K. Gregor, D. J. Rezende, F. Besse, Y. Wu, H. Merzic, and A. van den Oord, "Shaping Belief States with Generative Environment Models for RL," no. NeurIPS, 2019.

[7] D. Hafner, T. Lillicrap, I. Fischer, R. Villegas, D. Ha, H. Lee, and J. Davidson, "Learning latent dynamics for planning from pixels," *36th International Conference on Machine Learning, ICML 2019*, vol. 2019-June, pp. 4528–4547, 2019.

[8] D. Hafner, T. Lillicrap, J. Ba, and M. Norouzi, "Dream to Control: Learning Behaviors by Latent Imagination," pp. 1–20, dec 2019.

[9] L. Buesing, T. Weber, S. Racaniere, S. M. A. Eslami, D. Rezende, D. P. Reichert, F. Viola, F. Besse, K. Gregor, D. Hassabis, and D. Wierstra, "Learning and Querying Fast Generative Models for Reinforcement Learning," 2018.

[10] Ł. Kaiser and I. Sutskever, "Neural GPUs learn algorithms," *4th International Conference on Learning Representations, ICLR 2016 - Conference Track Proceedings*, pp. 1–9, 2016.

[11] C. Bamford and S. Lucas, "Neural Game Engine: Accurate learning of generalizable forward models from pixels," 2020.

[12] 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 *2019 IEEE Conference on Games (CoG)*. IEEE, aug 2019, pp. 1–8.

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

[14] 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.

[15] H. Tong, Y. Tao, and J. Liu, "Ppsn 2020 & ieee cog 2020 - gvgai learning competition," 2020. [Online]. Available: http://www.aingames.cn/gvgai/ppsn_cog2020

[16] 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.

[17] 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.

[18] A. Dockhorn, "Github repository: Corresponding files," 2020. [Online]. Available: https://github.com/ADockhorn/Local-Forward-Model-Learning-for-GVGAI-Games

[19] 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.

[20] V. Mnih, A. P. Badia, L. Mirza, A. Graves, T. Harley, T. P. Lillicrap, D. Silver, and K. Kavukcuoglu, "Asynchronous methods for deep reinforcement learning," *33rd International Conference on Machine Learning, ICML 2016*, vol. 4, pp. 2850–2869, 2016.

[21] J. Schulman, F. Wolski, P. Dhariwal, A. Radford, and O. Klimov, "Proximal Policy Optimization Algorithms," pp. 1–12, 2017.

[22] 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.

[23] 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.

[24] G. *SUSTechGameAI*, "Github repository: Gvgai learning competition in 2020," 2020. [Online]. Available: https://github.com/SUSTechGameAI/GVGAI_GYM