GDAŃSK UNIVERSITY OF TECHNOLOGY

TASK Quarterly 28 (4) 2024

Interpreting Deep Q-Networks: A Rule-Based Comparison with First-Order Logic in Wumpus World

Filip Pawlicki, Kamil Dobies, Marcin Pucek, Karol Draszawka *

Faculty of Electronics, Telecommunications and Informatics, Gdańsk University of Technology, Narutowicza 11/12, Gdańsk, 80-233, Poland
*karol.draszawka@pg.edu.pl

https://github.com/filippawlicki/Wumpus-Extracting-Logic

https://doi.org/10.34808/tq2024/28.4/c

Abstract

Deep reinforcement learning models such as Deep Q-Networks (DQNs) have achieved great performance in both simple and complex environments, but their decision-making process remains largely opaque. This work addresses the interpretability challenge by proposing a method of extracting and comparing symbolic rules from a trained DQN and logic-based agents. The method is showcased in the popular Wumpus World domain. Rules extraction from the agents is done via training decision trees to mimic the agent's behavior. Comparison is done using Jaccard similarity and simple structural metrics. Results show that despite similar performance, DQNs and logic agents rely on partially overlapping but structurally largely distinct decision rules. This highlights the feasibility of translating subsymbolic policies into interpretable rules and reveals meaningful structural differences between learned and symbolic strategies.

Keywords:

Explainable AI, Deep Q-Network, First-Order Logic

1. Introduction

Deep reinforcement learning (DRL) has achieved tremendous success in the treatment of sequential decision-making problems. One of the algorithmic milestones that has allowed this to occur is the development of the Deep Q-Network (DQN) family of solutions [1–3]. DQNs achieve outstanding performance in a wide variety of environments, from simple grid-based experiments to difficult tasks such as Atari games. The relevance of this algorithm is confirmed by the fact that its ideas are further developed in later deep reinforcement learning algorithms such as DDPG [4], SAC [5] or even MuZero [6].

However, although these DRL algorithms achieve great performance in a variety of tasks, they all share a common disadvantage: they are extremely opaque for a human interpreter that tries to understand their behavior, raising warning issues wherever interpretability, transparency, and safety aspects of methods are very important. The decision-making process of the DRL agents is encoded in the form of numerical weights and activations, making it difficult for humans to understand why a specific action was taken.

In contrast, symbolic agents, i.e. those with rules-based policies expressed in the language of First-Order Logic (FOL) – FOL agents – offer inherently interpretable decision processes. This contrast motivates our core research question: *Can the policy of a deep reinforcement learning agent, such as the DQN, be represented as a set of symbolic rules comparable to those defined by a logic-based agent?* If so, what are the similarities and differences in their internal reasoning patterns?

This work aims to make the behavior of DQN agents more interpretable by approximating their policies by learned symbolic rules and comparing these rules with those used by a logic-based agent. The contributions of this work are as follows.

- A comparison methodology between DQN and FOL agents is proposed based on: (1) Jaccard similarity to identify partially overlapping rules of their transformed transparent policies, and (2) the structural analysis of the extracted rules in terms of quantity, depth, and logical similarity.
- Four different agents (one FOL-based and three different DQN-based) operating in a Wumpus World under the same sensory input have been prepared and their policies have been transformed into explicit decision rules using decision trees trained to mimic their behavior.
- Using the proposed methodology, an empirical evaluation and comparison of the four agents is presented, which uncovers their key similarities and differences.

The findings suggest that, although the DQN and FOL agents achieve similar success rates, the underlying reasoning captured by their rule sets is only partially aligned. This indicates that subsymbolic (i.e., neural network-based) agents may arrive at effective policies through structurally different paths compared to explicitly programmed symbolic logic.

The remainder of this paper is structured as follows. The next section describes some of the previous research work related to explainable reinforcement learning and FOL agents. Section 3 presents the proposed method of interpretable comparison between agents. Section 4 describes the environment – Wumpus World – as well as the four agents developed to act in this environment. In Section 5, using the proposed method, logical rules for all agents are extracted and, based on this, they are compared and analyzed. A discussion about the presented method and results is provided in Section 6, including possibilities and limitations of the method. Finally, the last section concludes the findings and presents possible future extensions of this work.

2. Related Work

Explainable Reinforcement Learning (XRL) focuses on increasing the transparency of DRL agents by making their decision-making processes understandable to humans. XRL belongs to the more general area of explainable Artificial Intelligence (XAI). Both are topics that are highly researched. The broad range of methods developed in these fields, surveyed in [7–11], fall into two main categories.

One group of methods relies on writing *logical agents* or training only controller models that are inherently interpretable (e.g., Decision Trees (DTs)), thus building systems that provide intrinsic explanations. The classic approach to constructing logical agents is to use First-Order Logic (FOL) and logic programming in languages such as Prolog [12], Golog [13] and similar [14]. FOL provides a formal, interpretable framework for representing knowledge and reasoning. In the context of logical agents, FOL enables explicit modeling of the environment and decision-making rules, often leading to fully explainable behaviors. Examples of such agents can be found, e.g., in [15–17].

The other group of techniques provides post-hoc explanations to models, which are otherwise opaque to human interpretation, in particular neural network models. These methods can be either model specific or model agnostic. They can provide global explanations, that is, describing the full process of a response generation provided a given input, or give only a local, partial clue for an interpreter. Popular saliency maps (e.g., [18]) used for models

with visual input which show where the focus of a model is located in a given image can be classified as a post-hoc, model specific, local explanation method. Linear Interpretable Model-agnostic Explanations (LIME) [19] and SHapley Additive exPlanations (SHAP) [20] methods are popular model agnostic post-hoc local explanation methods which work for any predictors, but are especially useful for tabular data. The examples of global post-hoc explanation methods for neural network-based agents include building interpretable policy summarizations [21, 22], causal models [23–25] and symbolic approximations through e.g. genetic programming [26], Neurally Directed Program Search [27], converting neural activations into logical rules [28] or query-based rule extraction into Horn logic [29].

Finally, a symbolic approximation of a neural network can be obtained, as in this work, by training a decision tree to *mimic* the original neural network-based controller [30]. Such surrogate models provide human-understandable decision paths and enable analysis of the learned policy structure. As in this work, these decision paths of a decision tree can be converted into a set of logical rules.

The testing environment used in the work – Wumpus World – is a simple grid-based toy world, popularized by [31], where it was used to illustrate concepts such as logical reasoning, knowledge representation, and decision making under uncertainty, which has to be considered to develop a successful logical agent. There are many published solutions implementing agents for various versions of Wumpus World, both FOL-based [17, 32, 33] and DRL-based [34–36].

Although DTs have been used to interpret the behavior of DQN controllers before, e.g., in [37, 38], this has not been done for agents operating in the Wumpus World. Moreover, to our best knowledge, in no prior work have DQN controllers been compared against each other and with an FOL controller based on the generated rule sets.

3. Rule Set-Based Comparison of Agents

The proposed method of interpretable comparison of agents relies on transforming the original controllers (either written in an FOL language or trained using a DRL algorithm, like DQN) into sets of logic rules. These rule sets are then compared using the proposed structural metrics to obtain quantitative results. The rule sets also allow for qualitative analysis of each agent individually, as well as a detailed assessment of the similarities and differences in the decision rules of the agents.

3.1. Obtaining Rule Sets

To analyze the strategies learned by the DQN and First-Order Logic (FOL) agents in an interpretable way, the first step is to construct decision trees that mimic their behavior, which are later transformed into rule sets. In order to train decision trees, training data has to be collected. This is done as follows:

- each agent is run for N independent episodes a given environment – same for all agents;
- for every agent, each decision (i.e., each **observation** $\rightarrow$ **action** pair), is recorded; each episode provides a *trajectory* T of such training pairs;
- as a result, a dataset containing $N = M * \text{avg\_len}(T)$ such decision examples, with the exact number varying depending on the lengths of individual episodes (some agents have shorter average length of episodes than others).

Using such datasets, a separate decision tree for each agent is trained. The training is done in a standard way. Each dataset is split into training and test subsets using an 80/20 split. The input features consists of binary indicators representing the agent's perception of the environment, while action labels representing all possible agent decisions are treated as class indicators. The trees are trained using Gini impurity as the splitting criterion, with a maximum depth of `36` and a minimum of `10` samples per leaf node. These hyperparameters are chosen to balance accuracy and interpretability. To further simplify the models, a custom pruning algorithm is applied, which substitutes subtrees with identical class predictions in both children with a leaf node with that class prediction.

After training, the decision trees are transformed into rule sets. A single decision rule is represented as a conjunction of conditions imposed by each node along a root-to-leaf path in the decision tree. Extracted rules are expressed using standard logical notation. The following symbols are used throughout:

$\neg$: Negation (NOT)
$\wedge$: Conjunction (AND)
$\rightarrow$: Implication (IF ... THEN)
$\forall_s$: Universal quantifier (FOR ALL s)

For example, a rule could be represented as:

$$\forall_s \neg stench(s) \wedge glitter(s) \rightarrow \mathbf{grab}(\mathbf{s}).$$

This notation should be interpreted as: "For all fields (s), if the stench is not perceived in (s) and the glitter is perceived, then the agent performs the `grab` action."

Each path from the root node to a leaf node constitutes a single decision rule, as above.

3.2. Comparison of Rule Sets

To compare the extracted trees and their encoded decision rules, several structural metrics are calculated:

- **Number of Rules**: Each decision rule corresponds to a path from the tree root to a leaf. The total number of such paths indicates the complexity of the policy representation.
- **Maximum Rule complexity**: The longest path from root to any leaf, reflecting the deepest decision condition.
- **Average Rule complexity**: The mean length of all root-to-leaf paths, providing insight into the typical complexity of individual rules.

In addition to these structural metrics, decision rules are semantically compared by evaluating their similarity in the following way. For each pair of rules – one from the first agent and one from the second – the Jaccard similarity coefficient [39], a measure used for comparison between sets, is computed:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}, \tag{1}$$

where A and B denote the sets of conditions in the two compared rules.

Rules were considered to *match*, if their Jaccard similarity was at least 0.6. This threshold means that at least 60% of conditions overlapped, allowing us to identify logically similar, though not syntactically identical, decision strategies. Note that a single rule from one tree may be similar to multiple rules in the other tree due to this matching criterion. This approach enables detection of partially convergent policies between agents.

4. Showcase: Wumpus World Agents

4.1. The Environment

The environment consists of an $N \times N$ (4×4 in a classic version of the problem) grid containing several pits (three in the classic version), a Wumpus (the deadly monster), a gold bar, and an agent. Gold emits glitter, the Wumpus produces stench in adjacent cells, and pits create breeze in adjacent cells. The agent starts in the top-left corner of the grid, which is also the exit point after collecting the gold. The agent's goal is to collect gold and successfully escape without dying. Entering a cell with a pit or the Wumpus, or exceeding the maximum number of steps, results in the agent's death and failure.

The agent perceives the following basic sensory inputs under the following conditions:

- **Stench** – a Wumpus is in an adjacent cell,
- **Breeze** – a pit is in an adjacent cell,
- **Glitter** – gold is present in the current cell,
- **Bump** – the agent has hit a wall,
- **Scream** – the Wumpus was killed as a result of the previous action.

Additionally, the agent maintains internal knowledge of whether it is holding the gold (*has_gold*), whether it is at the entrance (*on_entrance*), and its current position (x,y).

The agent can perform the following actions:

- **Move Forward** – move one cell forward in the direction it is facing,
- **Turn Left / Turn Right** – rotate 90° in the corresponding direction,
- **Grab** – pick up the gold (only successful in the same cell as gold),
- **Climb** – exit the grid (only successful in the entrance cell),
- **Shoot** – fire an arrow (only successful once per game).

The agent always starts in the top-left corner of the grid. Other entities (pits, Wumpus, gold) are randomly placed, with the restriction that no hazards may be placed within the initial 2×2 area in the top-left corner, in order to reduce the number of unsolvable maps (when there is no path between the agent and gold). Each map is tested for the existence of a path to the gold, and if no such path exists, it is marked as unsolvable. The environment is rendered using the `pygame` library. The agent is represented by an arrow indicating its orientation; *G* denotes gold, *P* denotes a pit, and *W* represents the Wumpus. An example of a generated world is presented in Figure 1.

Figure 1: An example of a generated Wumpus World map. The explanation is in the main text.

- Exactly one Wumpus is placed on the map,
- Exactly one gold tile is present,
- All border tiles are surrounded by walls (the agent can not leave the map),
- The starting tile contains only the agent,
- There is a path from the starting tile to the gold (the gold is not blocked completely by pits).

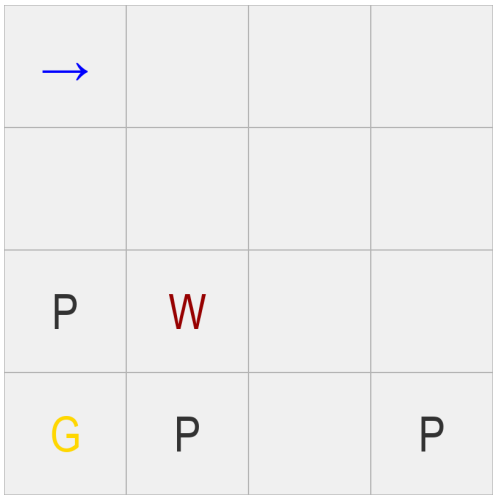

Figure 2: An example of an unsolvable Wumpus World map. The explanation is in the main text.

The example shown in Figure 2 illustrates an unsolvable map, where the gold is surrounded by pits, making it impossible for the agent to complete the game successfully.

4.2. The Agents

To showcase the rule-based comparison method, four agents operating in Wumpus World have been developed, including a symbolic agent based on first-order logic and reinforcement learning agents trained using a DQN algorithm. Each agent operates under the same perceptual constraints. The input state for the agent includes:

- Perceptual indicators (stench, breeze, glitter, bump, scream),
- Agent's position (x, y),
- Indicator whether the agent holds the gold,
- Indicator whether the agent is in the starting cell.

The action space is also identical for all the agents and consists of the actions: *Move forward*, *Turn left*, *Turn right*, *Grab*, *Climb*, and *Shoot*.

Although all agents have the same sensory input and the same action space, they rely on fundamentally different decision-making mechanisms.

4.2.1 FOL Agent

The First-Order Logic (FOL) agent utilizes symbolic reasoning and a dynamic knowledge base. It ensures safe, deterministic exploration and goal-oriented behavior in the Wumpus World. The logic-based reasoning is implemented using Prolog and then integrated into Python via the pyswip library.

Besides the perceptual input, the agent is also aware of its current relative position and orientation, as well as the step number S of the current state in the episode.

To facilitate decision-making, the agent stores the history of visited cells using the predicate `agent(X, Y, S)`, where (X, Y) denotes the position and S refers to the state number. Executed actions are memorized as `result(A, S)`, where A is an action taken in the state S.

The *knowledge base* is dynamically updated throughout the agent's exploration and is encoded using facts of the form `kb(X, Y, F, V)`, where:

- F is a cell property: *stench*, *breeze*, *glitter*, *visited*, *wumpus*, *pit*, *wall*, or *start*;
- V is the property's state: *unknown*, *false*, *possible*, or *true*.

Several auxiliary predicates are derived from the knowledge base to support the agent's inference and decision-making processes:

- `holding(gold, S)` – true if the agent holds the gold in state S;
- `has_shot(S)` – true if the agent has used its arrow;
- `cell_ahead(X, Y, Dir, NX, NY)` – identifies the cell (NX, NY) ahead of the agent, given the orientation *Dir*;
- `wumpus_targeted(X, Y, Dir)` – true if it is certain that the Wumpus is located in front of the agent;
- `valid_cell(X, Y)` – true if the cell (X, Y) is not a wall;
- `adjacent(X, Y, AX, AY)` - returns set of fields (AX, AY) adjacent to field (X, Y)
- field classification predicates: `visited_cell(X, Y)`, `safe_cell(X, Y)`, `good_cell(X, Y)`, `medium_cell(X, Y)`, `risky_cell(X, Y)`, and `deadly_cell(X, Y)`.

The agent applies the following inference rules to update its knowledge base:

- Detection of *stench*/*breeze* at (X, Y) implies the *possible* value of the Wumpus/pit in adjacent cells; its absence implies the *false* value;
- If *stench*/*breeze* is detected and only one neighbor has *possible* value for the Wumpus/pit, the other fields are set to false, and the remaining one is promoted to true (elimination method);
- If the agent occupies field (X, Y), then `kb(X, Y, wall, false)`;

- A *bump* implies that the cell in front is a wall;
- A *scream* implies that all cells are updated with *false* value for the Wumpus.

To structure decision-making, the agent evaluates candidate actions within a five-level utility hierarchy:

- **great**: exiting the cave, grabbing gold, or shooting with certainty about Wumpus location;
- **good**: returning to a visited cell if the agent has grabbed the gold or moving to a safe unvisited cell otherwise;
- **medium**: revisiting a previously visited cell;
- **risky**: moving to a cell with *possible* state for the Wumpus or the pit;
- **deadly**: moving to a cell marked *true* for the Wumpus/pit.

For each state S, the agent retrieves relevant information from predicates like `agent(X, Y, S)`, `orientation(Dir, S)`, `cell_ahead(X, Y, Dir, NX, NY)` and `adjacent(X, Y, AX, AY)`. It then applies an if-else rule chain, where each condition is evaluated in order until a matching action is selected:

$$\begin{aligned}
\texttt{holding(gold, S)} \wedge \texttt{kb(X, Y, start, true)} &\Rightarrow \texttt{climb},\\
\texttt{kb(X, Y, glitter, true)} &\Rightarrow \texttt{grab},\\
\texttt{wumpus_targeted(X, Y, Dir)} &\Rightarrow \texttt{shoot},\\
\neg\texttt{valid_cell(NX, NY)} &\Rightarrow \texttt{turn},\\
\texttt{good_cell(NX, NY)} &\Rightarrow \texttt{move},\\
\exists_{AX,\,AY}\ \texttt{good_cell(AX, AY)} &\Rightarrow \texttt{turn},\\
\texttt{medium_cell(NX, NY)} &\Rightarrow \texttt{move},\\
\exists_{AX,\,AY}\ \texttt{medium_cell(AX, AY)} &\Rightarrow \texttt{turn},\\
\texttt{risky_cell(NX, NY)} &\Rightarrow \texttt{move},\\
\exists_{AX,\,AY}\ \texttt{risky_cell(AX, AY)} &\Rightarrow \texttt{turn},\\
\texttt{deadly_cell(NX, NY)} &\Rightarrow \texttt{move},\\
\text{else} &\Rightarrow \texttt{turn}.
\end{aligned}$$

Expressions such as $\exists_{AX,\,AY}$ `[ ]_cell(AX, AY)` are true if there exists at least one cell (AX, AY) adjacent to the agent that satisfies the given classification predicate. The agent then turns toward the nearest such cell; if the cell lies behind its current orientation, the agent rotates left to gradually reorient.

The agent is designed to be more general than strictly required by the environment's constraints. In our specific environment, the rule sequence typically terminates at the conditions corresponding to finding a medium cell. The remaining predicates are only relevant in situations where the knowledge base contains no safe cells—such as when the agent perceives a breeze or stench on the starting field. Nonetheless, our environment is deliberately configured to prevent pits and Wumpuses from being placed adjacent to the starting location.

4.2.2 Deep Q-Network Agents

For the purpose of this study, three variants of agents using the Deep Q-Network (DQN) method have been trained. These agents differ mainly in their reward allocation mechanisms and use of memory components. Each of them was obtained with the use of DQN adapted to the Wumpus World environment as presented in [36] – including their two-stage epsilon-greedy strategy for action selection (two independent sets of exploration parameters: the first set for states until the gold is picked up – it allows the agent to intensively explore the environment and learn to improve in finding the gold; and the second set for states from the moment the gold is acquired until the end of the episode – encouraging learning an effective return strategy).

A Deep Q-Network (DQN) is a reinforcement learning algorithm that combines classical Q-learning with neural networks to approximate the Q-value function. The Q-Network works as a function $Q(s,a)$, which estimates the expected cumulative reward for performing action a in state s and thereafter following an optimal policy. The network input is a state vector of size `input_dim` containing the agent's perception, position, and state. The output is a vector of Q-values for each possible action.

The implemented DQN model consists of three fully connected layers with ReLU activation functions [40] between them. The network architecture is summarized in Table 1.

Table 1: Neural network architecture of the DQN model.

Layer	Type	Input Size	Output Size
fc1	Linear	input_dim	256
ReLU	Activation	256	256
fc2	Linear	256	256
ReLU	Activation	256	256
fc3	Linear	256	output_dim

The complete set of training hyperparameters used in the experiments is presented in Table 2.

Table 2: Training parameters for the DQN model.

Parameter	Value
Loss function	SmoothL1Loss [41]
Optimizer	Adam [42]
Learning rate	10^{-3}
Discount factor (γ)	0.9
Replay memory size	100,000
Batch size	64
Exploration strategy	Two-stage ε-greedy [36]
Target network update frequency	every 25 episodes
$\varepsilon_{\text{start1}}$	1
$\varepsilon_{\text{decay1}}$	2×10^{-4}
$\varepsilon_{\text{min1}}$	0.1
$\varepsilon_{\text{start2}}$	1
$\varepsilon_{\text{decay2}}$	2×10^{-4}
$\varepsilon_{\text{min2}}$	0.1

These parameters were selected to balance training stability, convergence speed, and exploratory behavior across episodes.

Baseline DQN Agent The baseline DQN model follows a standard approach for training an agent in the Wumpus environment, where the agent receives rewards for basic events and actions during an episode.

The reward function for the baseline model is as follows:

- **Movement penalty:** Each agent step incurs a penalty of -1, encouraging the agent to minimize the number of actions and reach the goal efficiently.
- **Wall collision:** An attempt to move into a wall results in a -5 penalty, with no change in the agent's position.
- **Gold acquisition:** Picking up gold yields a large reward of $+500$, strongly motivating the agent to find and retrieve the gold.
- **Invalid actions:** Actions such as attempting to pick up gold when not on its tile, shooting without an arrow, or climbing without meeting the conditions incur a penalty of -20.
- **Exploring new tiles:** Visiting a previously unvisited tile grants a reward of $+50$, encouraging exploration.
- **Revisiting tiles:** Re-visiting a tile results in a small penalty of -0.001 to prioritize exploration.
- **Killing the Wumpus:** Successfully hitting the Wumpus with an arrow yields a reward of $+300$.
- **Escaping the cave:** Leaving the cave with gold grants the highest reward of $+1000$ and ends the episode.
- **Agent death:** Entering a tile with a live Wumpus or a pit leads to immediate death and a -1000 penalty, ending the episode.
- **Step limit exceeded:** Surpassing the limit of 100 steps ends the episode with a -1000 penalty.

The reward function is designed to reflect the task's essence: finding and retrieving the gold and safely exiting the cave, while penalizing redundant and incorrect behaviors.

Explorative DQN Agent The second agent shares the same DQN architecture as the baseline model, but employs significantly altered reward functions. The proposed reward modifications aim to encourage exploration and safe return with the gold while reducing unproductive behaviors. The changes to the reward system includes:

- **Penalty for revisiting tiles:** Increased to -20 to discourage aimless movement.
- **Reward for visiting new tiles:** Increased to $+120$ to promote exploration of unknown areas.
- **Reward for revisiting tiles after collecting gold:** $+20$ to encourage returning via safe, known routes.
- **Wall collision penalty:** Increased to -50 to reduce behavior near walls when facing threats.
- **Death penalty:** Reduced to -100 to encourage risk-taking in exploration.
- **Winning reward:** Increased to $+10,000$ to clearly prioritize success over mere survival.

Such altering of the reward signals significantly changed agent behavior - rather than choosing safe but ineffective actions (like bumping into walls), the agent makes riskier yet task-relevant decisions. The increased reward for exploration and decreased penalty for death leads to more dynamic behavior and more efficient strategies for finding gold and exiting the cave.

DQN Agent with Memory The third agent builds upon the *explorative* model, maintaining its modified reward function, but adds an internal memory mechanism based on sensory maps to enhance its cognitive abilities.

Following [36], the agent's cognitive capabilities were extended with three separate *sensation maps*, enabling the agent to store and update knowledge based on past observations:

- **Stench map:** A $M \times M$ grid tracking *stench*, which indicate proximity to the Wumpus.
- **Breeze map:** A $M \times M$ grid tracking *breeze*, which indicate proximity to pits.
- **Wall map:** A larger $(M+2) \times (M+2)$ grid covering possible movement areas and the cave's boundary, storing wall collisions (bumps) and the agent's own position.

Here, $M = 2 \cdot grid\_size - 1$, with the wall map including the cave perimeter. The agent always starts in the center of the map, reflecting no prior knowledge of its actual location within the cave.

Map update rules are as follows:

- After each move, if a *stench* or *breeze* is detected, the corresponding map entry is set to $+1$; absence is marked as -1.
- In the wall map, a bump results in a $+1$ on the tile the agent faces, while the agent's current position is marked with -1 (updated each turn).

The agent's memory is enriched over time, and the current state of these maps is appended to its observations. For example, with a grid size of 4 (i.e., $M = 7$), the final input vector to the neural network includes the original observations plus:

$$7 \times 7 + 7 \times 7 + 9 \times 9 = 49 + 49 + 81 = 179$$

additional input features.

All other model hyperparameters remained unchanged.

This memory system allows the agent to better understand the environment's structure and make informed decisions based on prior experiences, enabling more strategic planning and efficient exploration.

5. Experimental Results

This section presents the analysis of how the extracted decision trees reflect the underlying strategies and to what extent the learned rules align with the symbolic logic encoded in the FOL agent. Before this comparison, each agent's direct test results are briefly presented.

5.1. Performance Evaluation

All of the agents - FOL, Baseline DQN, Explorative DQN and Memory DQN - were tested across 10,000 randomly generated games. The map set was the same for each agent. For each model, the following metrics were recorded:

- **Win ratio** – percentage of games where the agent successfully retrieved the gold and escaped,
- **Survival rate** – percentage of games where the agent survived (was not killed by the Wumpus or fell into a pit),
- **Average steps** - average amount of steps that the agent needed to win,
- **Max steps** - the number of steps taken in the longest game won,
- **Unwinnable games** - percentage of games with configurations that made winning impossible (e.g., gold surrounded by pits).

Table 3: Comparison of agent performance in the Wumpus world (10,000 test episodes).

Unsolvable games: 1.87%				
Model	**Win ratio**	**Survival**	**Avg steps**	**Max steps**
Baseline	31.54%	71.74%	12.85	31
Explorative	38.03%	78.98%	13.99	34
Memory	45.22%	85.15%	15.39	38
FOL	50.75%	100.0%	22.85	88

As shown in Table 3, each successive modification of the agent led to a significant improvement in performance and to a progressively closer alignment with the results of the FOL agent. The memory-based agent achieved the best results among the DQN models, demonstrating superior perception and internal environment representation, which enabled more effective decision-making in an environment with limited observability.

Out of all simulations, 1.87% of the worlds were found to be unsolvable - that is, they contained no path to the gold and back at all.

The FOL agent achieved a survival rate of 100% and a win rate of 50.75%. On average, it required 22.85 steps to complete a winning game, with a maximum of 88 steps observed. Although simulations were conducted with higher step limits, it was found that a cap of 100 steps was sufficient, as no agent had ever succeeded beyond that threshold.

In roughly half of the games, the agent failed to win due to insufficient knowledge - no adjacent cell could be classified as safe with certainty anymore. In such cases, it would loop between previously visited cells. For every agent, games that ended with loop, were terminated after reaching the 100-step limit.

5.2. Rule Sets Extraction

Using the proposed method presented in Sec. 3, for each of the presented agents, a decision tree mimicking its behavior was trained. Each tree achieved approximately 70% classification accuracy on their respective test sets, demonstrating a reasonable fit of the surrogate models to the agents' decision policies.

5.2.1 FOL Agent Rule Set

It is important to note that only the agent's direct observations were passed to the tree and not the knowledge base. The rules obtained from the decision tree for the FOL agent are presented in Listing 1.

Transforming the decision tree into this set of logical rules allows for straightforward comparison with the principles that emerged in the DQN models during training.

The most frequent action appearing in the tree is

Turn. The clearest example is at the top of the tree - whenever a *bump* is detected, the agent immediately turns. In second place, despite *Grab* appearing in more leaf nodes, is *Move Forward*. Although it appears in only one leaf, it represents almost the largest group of cases. This indicates the agent's caution in making risky decisions - it moves forward mainly when it senses no threat (in the tree, when *stench* and *breeze* are both low). *Move Forward* also appeared in other situations but was not the dominant action, which shows that when the agent sensed risk, it tended to turn toward a safer square rather than moving forward immediately.

It is also worth noting the correctness of the *Grab* and *Climb* actions. When the agent detected *glitter*, it immediately picked up the gold, and when it was on the entrance tile while holding the gold, it immediately performed the *Climb* action. The agent was decisive in this regard - there are no other actions mixed in the *Grab* leaves, and the same applies to the *Climb* leaves. These actions also do not appear elsewhere in the tree.

One can also notice the absence of a leaf with the *Shoot* class, as well as the lack of a node with the *scream* criterion. This is because the *Shoot* action occurred too rarely compared to other actions under the given conditions to become dominant. The conditions for executing a *Shoot* were quite specific and required the agent's experience and confidence about the Wumpus's location. The absence of the *scream* condition suggests that this percept had negligible direct impact on the agent's behavior. Instead, it influenced the strategy indirectly: the agent would only shoot when confident of hitting the target, which always produced a *scream*. After shooting, the tiles where *stench* was previously perceived would lose that percept, and this change in *stench* was the decisive factor that guided the agent's future actions.

Listing 1: The rule set obtained for the FOL agent.

$$\forall_s \neg bump(s) \wedge breeze(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge \neg on\_entrance(s) \wedge stench(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge \neg on\_entrance(s) \wedge \neg stench(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge on\_entrance(s) \wedge has\_gold(s) \rightarrow \mathbf{climb(s)}$$

$$\forall_s bump(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s \neg bump(s) \wedge breeze(s) \wedge \neg glitter(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge on\_entrance(s) \wedge \neg has\_gold(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge \neg on\_entrance(s) \wedge stench(s) \wedge \neg glitter(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s \neg bump(s) \wedge \neg breeze(s) \wedge \neg on\_entrance(s) \wedge \neg stench(s) \wedge \neg glitter(s) \rightarrow \mathbf{move(s)}$$

5.2.2 Baseline DQN Agent Decision Tree

The rules extracted from the decision tree for the baseline DQN agent are presented in Listing 2.

Listing 2: The rule set obtained for the basic DQN agent.

$$\forall_s \neg stench(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s stench(s) \wedge \neg bump(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s \neg stench(s) \wedge \neg glitter(s) \wedge \neg breeze(s) \wedge has\_gold(s) \wedge on\_entrance(s) \rightarrow \mathbf{climb(s)}$$

$$\forall_s \neg stench(s) \wedge \neg glitter(s) \wedge breeze(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s stench(s) \wedge \neg bump(s) \wedge \neg glitter(s) \wedge breeze(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s stench(s) \wedge bump(s) \wedge \neg breeze(s) \wedge \neg has\_gold(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s stench(s) \wedge \neg bump(s) \wedge \neg glitter(s) \wedge \neg breeze(s) \wedge \neg has\_gold(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s stench(s) \wedge bump(s) \wedge breeze(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s stench(s) \wedge bump(s) \wedge \neg breeze(s) \wedge has\_gold(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s \neg stench(s) \wedge \neg glitter(s) \wedge \neg breeze(s) \wedge \neg has\_gold(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s stench(s) \wedge \neg bump(s) \wedge \neg glitter(s) \wedge \neg breeze(s) \wedge has\_gold(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s \neg stench(s) \wedge \neg glitter(s) \wedge \neg breeze(s) \wedge has\_gold(s) \wedge \neg on\_entrance(s) \rightarrow \mathbf{move(s)}$$

The rules show that the agent uses straightforward perceptual conditions to decide its next action. The dominant actions in the extracted tree are *Move Forward* and *Turn* - this suggests that the agent focuses on exploring the cave, as expected. When perceiving observations such as *stench* or *breeze*, the agent chooses to turn rather than risk stepping onto a dangerous tile. In safer states, when no risk factors are detected, moving forward becomes the preferred action.

The actions *Grab* and *Climb* appear in clearly defined situations: the agent grabs the gold when *glitter* is present and climbs out of the cave when it has the gold and is on the entrance tile. That shows that the decision tree correctly reflects expected behaviors for collecting gold and exiting the environment.

The *Shoot* action does not appear in the decision tree at all. This may suggest that the agent has learned to play without using its arrow, possibly due to the penalty for missing, or that it reliably finds a path to the gold without the need of eliminating the Wumpus. The presence of the *scream* percept in the data implies that *Shoot* must have occurred during gameplay, but the low frequency or dispersed context likely prevented it from forming a dominant branch in the tree.

Despite the generally correct strategies, the tree also reveals some suboptimal behaviors. For example, when a *bump* occurs, in only 83 out of 2030 cases does the agent choose to turn; in the remaining cases, it moves forward, hitting a wall. This may suggest that the agent prefers a minor penalty for bumping into a wall rather than risking a turn into an unknown direction, which could lead into a pit and receiving a higher penalty.

5.2.3 DQN Explorative Agent Decision Tree

The rules extracted from the decision tree mimicking the explorative DQN agent are presented in Listing 3.

The decision tree created for the *DQN Explorative* agent reveals a very similar structure to that for the previous DQN model, despite slight differences in the order of branches. The dominant actions remain *Move Forward* and *Turn*. When the agent perceives potential threats (i.e., when *stench* or *breeze* is noticed), it still tends to turn rather than move forward, avoiding risky tiles. Notably, the *bump* perception does not appear in the extracted rules, which may indicate that the agent learned to avoid collisions due to the higher penalty for hitting a wall.

The *Grab* and *Climb* actions again appear in expected contexts. Unlike the previous trees, this one also includes the *Shoot* action, triggered when both *stench* and *breeze* are perceived without gold being carried. Furthermore, the presence of the *scream* percept shows that the agent occasionally succeeds in hitting the Wumpus.

Listing 3: The rule set obtained for the Explorative DQN agent.

$$\forall_s\ breeze(s) \land stench(s) \land glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ breeze(s) \land \neg stench(s) \land glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \land \neg on\_entrance(s) \land stench(s) \land glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \land \neg on\_entrance(s) \land \neg stench(s) \land glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \land on\_entrance(s) \land has\_gold(s) \rightarrow \mathbf{climb(s)}$$

$$\forall_s\ breeze(s) \land stench(s) \land \neg glitter(s) \land \neg has\_gold(s) \rightarrow \mathbf{shoot(s)}$$

$$\forall_s\ breeze(s) \land \neg stench(s) \land \neg glitter(s) \land \neg scream(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ \neg breeze(s) \land \neg on\_entrance(s) \land stench(s) \land \neg glitter(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ \neg breeze(s) \land on\_entrance(s) \land \neg has\_gold(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ breeze(s) \land stench(s) \land \neg glitter(s) \land has\_gold(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ breeze(s) \land \neg stench(s) \land \neg glitter(s) \land scream(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ \neg breeze(s) \land \neg on\_entrance(s) \land \neg stench(s) \land \neg glitter(s) \rightarrow \mathbf{move(s)}$$

5.2.4 Decision Tree of the DQN Agent with Memory

The rules extracted from the decision tree that describes the DQN agent with memory are shown in Listing 4.

The decision tree of the DQN agent with memory reveals that the action *Turn* remains dominant, while the basic rules (picking up gold when *glitter* is perceived, climbing when *on_entrance* and *has_gold* are true) remain unchanged compared to previous models. The *Shoot* action does not appear explicitly, but the presence of the *scream* percept suggests that shooting occurred at least occasionally. Unlike the previous Explorative DQN tree, the *bump* perception is present here, and the reaction to it is *Move Forward*, meaning the agent sometimes attempts

to move into a wall. This likely reflects a rare edge case – it occurred in only about 900 out of all 200,000 cases – and might result from the increased complexity of learned conditions rather than a change in reward shaping.

Listing 4: The rule set obtained for the DQN agent with memory.

$$\forall_s\ breeze(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \wedge stench(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge \neg has\_gold(s) \wedge glitter(s) \rightarrow \mathbf{grab(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge has\_gold(s) \wedge on\_entrance(s) \rightarrow \mathbf{climb(s)}$$

$$\forall_s\ breeze(s) \wedge \neg glitter(s) \wedge has\_gold(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ breeze(s) \wedge \neg glitter(s) \wedge \neg has\_gold(s) \wedge \neg bump(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge \neg has\_gold(s) \wedge \neg glitter(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge has\_gold(s) \wedge \neg glitter(s) \wedge \neg bump(s) \wedge on\_entrance(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge has\_gold(s) \wedge \neg glitter(s) \wedge \neg bump(s) \wedge \neg on\_entrance(s) \wedge \neg scream(s) \rightarrow \mathbf{turn(s)}$$

$$\forall_s\ breeze(s) \wedge \neg glitter(s) \wedge \neg has\_gold(s) \wedge bump(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge has\_gold(s) \wedge \neg on\_entrance(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge \neg has\_gold(s) \wedge \neg glitter(s) \wedge bump(s) \rightarrow \mathbf{move(s)}$$

$$\forall_s\ \neg breeze(s) \wedge \neg stench(s) \wedge \neg has\_gold(s) \wedge \neg glitter(s) \wedge \neg bump(s) \wedge \neg on\_entrance(s) \wedge scream(s) \rightarrow \mathbf{move(s)}$$

5.3. Rule Sets Comparison

Table 4: Comparison of logical rules for the DQN agents with those for the FOL agent. The row *# matched with FOL* shows the number of rules that are matched with FOL agent using the Jaccard similarity method.

Feature	FOL	Basic DQN	Explorative DQN	Memory DQN
Number of rules	9	12	12	13
# matched with FOL	-	4	5	7
Avg rule complexity	3.89	3.92	3.67	4.31
Max rule complexity	5	5	4	7

Table 4 presents a comparison of the logical rules extracted for the FOL agent and all three DQN agents.

The *basic* agent generated more rules (12) than the FOL agent (9). Notably, four of the rules formed by the *basic* agent overlap with those of the FOL agent, indicating a partial similarity in decision-making. The average and maximum rule complexities for both agents are comparable, suggesting that the logical complexity of the rules (in terms of the number of conditions required to reach a decision) is similar. The maximum depth (five levels) is identical for both agents, indicating a comparable level of detail in their extracted strategies.

The *explorative* agent produced 12 rules - the same as the *basic* agent and more than the *FOL*-based agent, which produced 9. Notably, five of the *explorative* agent's rules overlap with those of the *FOL* agent, which indicates slightly greater similarity in decision-making compared to the *basic* agent, which shared four. The average rule complexity of the *explorative* agent's tree (3.67) is slightly lower than that of the *FOL* agent (3.89), suggesting that its rule structures are somewhat simpler. The maximum complexity is also lower (4 compared to 5), which may point to a less detailed representation of the decision logic. However, the higher number of shared rules indicates that the *explorative* agent more effectively reproduced some of the behavior of the symbolic agent.

The *memory-based* agent generated the highest number of rules among all analyzed models – 13 – 7 of which overlap with the rules of the FOL agent. This indicates the highest level of similarity and suggests strong alignment with the symbolic decision strategy. In terms of structural complexity, the *memory-based* agent has both the highest average (4.31) and maximum (7) complexity of rules, pointing to more nuanced logical rules and a higher level of detail in decision making compared to other agents and especially the FOL agent. This suggests that incorporating history and internal memory may contribute to learning richer and more precise rule sets.

5.4. Results Across Different Grid Sizes

To evaluate the scalability and generalization capabilities of the agents, additional experiments on maps of increasing size (4×4 to 7×7) were conducted. Comparison between the rules for the FOL-based agent and the best DQN agent (with memory maps) was made, as well as between their overall win rates, survival rates, and average steps to win.

As shown in Table 5, with increasing map sizes, a consistent growth in the number of extracted decision rules is observed, particularly for the memory-based model. The number of similar rules to those used by the FOL agent also increased with grid size, suggesting that the agent's behavior becomes more structured and rule-aligned in more complex environments.

Table 5: Comparison of agent rules similarity on different sized grids.

Grid Size	FOL rules	Mem.-based rules	Matched rules
4	7	15	7
5	11	16	12
6	13	17	15
7	13	17	16

As shown in Table 6, the FOL agent consistently achieved higher win rates and maintained 100% survival across all grid sizes. In contrast, the DQN agent showed decreasing win rates as grid size increased, but maintained relatively stable survival rates due to cautious behavior learned through negative reinforcement.

Table 6: Performance metrics for both agents across different grid sizes.

Metric	Size 4	Size 5	Size 6	Size 7
Memory Agent				
Win Rate	43.66%	27.60%	19.36%	15.21%
Survival Rate	85.20%	86.25%	86.39%	87.69%
Avg Steps to Win	15.27	15.60	15.61	16.60
FOL Agent				
Win Rate	49.99%	55.58%	60.53%	64.43%
Survival Rate	100.00%	100.00%	100.00%	100.00%
Avg Steps to Win	22.76	31.34	41.46	52.65
Unsolvable Maps	2.28%	0.34%	0.11%	0.03%

The achieved results demonstrate that while the logic-based FOL agent maintains superior and more consistent performance across all map sizes, the memory-based DQN agent exhibits increasingly structured behavior that better approximates symbolic rules as the environment becomes more complex. However, the DQN agent's win rate declines with larger maps, indicating that additional enhancements — such as better reward system or hybrid symbolic integration — may be needed for generalization to more challenging tasks.

5.5. Summary

The rules extracted for the *FOL* agent are clear and logically consistent. This agent behaves cautiously - it performs *turn* when it encounters a *bump* or any percept indicating danger, and the *move_forward* action is executed only when no warning signals (*breeze* or *stench*) are detected. It correctly interprets *glitter* as a signal to *grab*, and performs *climb* only when it is in the entrance field and carrying gold.

The *basic* agent generated 12 rules, of which 4 overlap with the FOL agent's rules. Although its rules are slightly more complex than those of the FOL agent, they are sometimes less intuitive - riskier actions appear, such as *move_forward* despite the presence of *stench* or *breeze*. The maximum tree depth (5) is identical to that of the FOL agent, indicating a comparable level of decision detail.

The *explorative* agent also produced 12 rules, but 5 overlap with the FOL agent's rules, suggesting greater alignment in decision making. Its average tree depth (3.67) and maximum depth (4) are both lower than the FOL agent's, implying simpler structures. However, the higher rule overlap may indicate more balanced training - better exploration helped the agent more closely approximate the logical decisions of the symbolic agent.

The most advanced is the *memory-based* agent, which generated 13 rules - the highest among all models - with as many as 7 overlapping with the FOL agent's rules. Its average tree depth is 4.31 and maximum depth is 7, indicating high structural complexity and context dependence. The memory component allows the agent to consider the history of percepts, resulting in more detailed and semantically aligned rules that are closer to the FOL strategy.

Importantly, as model complexity increases (from *basic* to *explorative* to *memory-based*), not only is higher rule overlap with the FOL agent observed, but also key performance indicators, win ratio and survival rate, get progressively better. The *basic* agent achieved a 31.26% win ratio and 71.78% survival rate; the *explorative* agent improved these to 37.32% and 78.57%, respectively; and the *memory-based* agent reached 42.33% and 87.67%.

This trend in rule alignment is also reflected in the overall performance evaluation conducted over 10,000 test games with identical randomly generated maps for all agents. The *memory-based* DQN model outperformed other learned agents in terms of win ratio (45.22%) and survival rate (85.15%), demonstrating improved perception and environment representation. The *FOL* agent maintained the highest performance, achieving a 50.75% win ratio and perfect survival (100%), although with more average steps to successful game completion. Approximately 1.87% of all games were unwinnable due to map configurations, affecting all agents equally.

These results highlight that increasing model complexity and memory capacity in learned agents not only brings their decision-making rules closer to the logically grounded FOL strategy but also leads to better practical performance in the Wumpus environment, suggesting that interpretability and effectiveness can be jointly improved.

In summary:

- The *FOL* agent serves as a benchmark for logical, safe, and consistent decision making.
- The *basic* agent optimizes the reward but its rules can be harder to interpret.
- The *explorative* agent demonstrates clearer rules and better alignment with the FOL agent.
- The *memory-based* agent generates the richest set of rules - deep and highly consistent with the FOL rules - highlighting the potential of memory to enhance interpretability in learning agents.
- Greater rule alignment goes hand in hand with improved performance metrics, suggesting that interpretability and efficiency can coexist.

6. Discussion

To this end, three variants of DQN agents were implemented: a baseline model, an explorative model with reward shaping, and a memory-based model utilizing internal sensory maps. While each version showed incremental improvements in performance, the primary goal of this work was not to maximize reward, but rather to analyze and interpret the decision-making strategies learned by neural agents. Particular attention was given to extracting the decision logic of DQN agents and comparing it with the rules used by a First-Order Logic (FOL) agent.

The proposed rule extraction method for DQN models, based on generating a large dataset of observations and decisions followed by decision tree training, enables an interpretable approximation of their behavior in a symbolic form. Although the extracted rules are not identical to the hand-coded rules used by the FOL agent, they show multiple similarities, especially in areas such as threat avoidance and reaching the exit after collecting gold.

At the same time, differences between the agents emerge due to the chosen reward functions and the limitations of the learning process itself. Additional experiments showed that reward shaping and adding memory influence the structure of the learned strategies and the complexity of the extracted rules. These modifications led to more consistent behaviors and increased similarity between DQN and FOL agent rules.

From the perspective of building general cognitive systems (e.g., robots navigating unknown environments), the ability to incrementally build knowledge and memory appears particularly important. The FOL agent relies on a successively updated knowledge base (KB) that explicitly and verifiably represents the state of the environment. This enables not only the explanation of decisions but also their formal verification and identification of situations where the agent gets "stuck" (e.g., when no safe fields are available).

In contrast, DQN agents acquire knowledge in a distributed and implicit manner, encoded within the neural network weights. Although this allows for adaptation and generalization, it lacks formal correctness guarantees and direct reasoning capabilities. Rule extraction therefore serves as a bridge between these two paradigms, making machine learning-based systems more transparent and understandable.

The conducted analysis also has limitations. The extracted rules are only an approximation of the DQN policy, resulting from both imperfections in the extraction process (e.g., limited classification accuracy of surrogate decision trees, around 70% on held-out data) and the fact that the agent's policy is not deterministic and may depend on complex, multidimensional input patterns.

This leads to a notable misalignment between the extracted rules and the true underlying DQN policies. Some of the rules visible in the surrogate may never be executed by the original agent, and conversely, some frequently used neural decision patterns may not appear in the extracted rule set at all. Consequently, the symbolic rules should be viewed as a partial and potentially biased snapshot of the learned policy rather than a faithful reconstruction, and apparent inconsistencies or oversimplifications in the rules may reflect limitations of the surrogate model rather than deficiencies of the agent itself.

Moreover, the analysis method relies on a static dataset and does not account for changes occurring during the agent's learning process. Apart from that, the applied rule similarity metric (Jaccard similarity) focuses only on condition overlap and does not consider the relative importance of individual features.

Future work could address these issues by employing ensembles of interpretable surrogates, probabilistic rule extraction, or online extraction during training, as well as by systematically evaluating surrogate fidelity to assess how much interpretability is being traded off against faithfulness to the agent's true decision process.

It is also worth noting that while the Wumpus World provides a convenient testbed due to its well-defined rules and symbolic structure, it remains a highly simplified and artificial environment. Scaling the proposed rule extraction approach to larger and more realistic domains would introduce challenges such as continuous state and action

spaces, high-dimensional sensory inputs, stochastic dynamics, and temporal abstraction.

In these settings, the extracted rules may need to incorporate probabilistic conditions or abstractions rather than purely symbolic propositions, and more expressive surrogate models than plain decision trees - for example, hierarchical trees, probabilistic rule lists, or hybrid symbolic-subsymbolic architectures - would be necessary to capture richer feature interactions without sacrificing interpretability. Another difficulty lies in long-term dependencies and delayed consequences, which require extracting not only single-step decision rules but also higher-level policies.

Addressing these issues could transform the proposed method into a general tool for explaining deep reinforcement learning policies far beyond grid-based settings, while still providing human-readable insights into how policies evolve in increasingly complex environments such as robotics or autonomous navigation.

7. Conclusion and Future Work

This study investigated the use of Deep Q-Networks (DQN) in the Wumpus World environment, with a focus on making agent behavior more interpretable to humans. The goal was to develop agents whose decisions could be better understood and potentially translated into symbolic or logic-based rules.

The memory-based agent, despite its complexity, proved especially useful for interpretability: it produced the largest set of decision rules, many of which closely matched the logical patterns observed in the FOL agent. This suggests that memory mechanisms can support not only more effective decision-making, but also greater alignment with interpretable, rule-based reasoning.

A key insight from this work is that the introduction of structured internal memory not only improved performance but also enhanced interpretability. The agent's sensory maps offered insight into how environmental features were perceived and stored over time, helping to analyze decision-making processes.

This paves the way toward more interpretable reinforcement learning (RL) systems, where agents are not treated as black boxes but rather as cognitive entities whose behavior can be reasoned about and debugged through their internal representations.

Future research could expand this approach to more complex or dynamic environments and incorporate probabilistic reasoning based on the agent's internal memory maps. Integrating symbolic reasoning or logic-based modules may further improve both interpretability and robustness.

Another promising direction involves exploring meta-learning techniques that enable agents to adapt their memory architecture or reward functions over time, allowing for more flexible and intelligent behavior. Looking ahead, hybrid systems that integrate symbolic reasoning with reinforcement learning hold significant potential. The goal would be to develop agents that not only learn effectively through interaction but also autonomously build and update an internal knowledge base and use it for planning actions. Such solutions have the potential to become the foundation of more general artificial intelligence systems.

Lastly, applying this method to environments with continuous action spaces or richer forms of partial observability would offer a valuable benchmark for assessing its generalization potential beyond grid-based domains.

Acknowledgements

We would like to sincerely thank Professor Julian Szymański for his scientific mentoring and valuable guidance during the preparation of this paper. His expertise in artificial intelligence and continuous support greatly contributed to the development and quality of this work.

References

[1] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," 2013.

[2] H. v. Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 2094–2100, AAAI Press, 2016.

[3] T. Schaul, J. Quan, I. Antonoglou, and D. Silver, "Prioritized experience replay," in *4th International Conference on Learning Representations, ICLR 2016, San Juan, Puerto Rico, May 2-4, 2016, Conference Track Proceedings* (Y. Bengio and Y. LeCun, eds.), 2016.

[4] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning.," in *ICLR* (Y. Bengio and Y. LeCun, eds.), 2016.

[5] T. Haarnoja, A. Zhou, P. Abbeel, and S. Levine, "Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor.," in *ICML* (J. G. Dy and A. Krause, eds.), vol. 80 of *Proceedings of Machine Learning Research*, pp. 1856–1865, PMLR, 2018.

[6] J. Schrittwieser, I. Antonoglou, T. Hubert, K. Simonyan, L. Sifre, S. Schmitt, A. Guez, E. Lockhart, D. Hassabis, T. Graepel, T. Lillicrap, and D. Silver, "Mastering Atari, Go, chess and shogi by planning with a learned model," *Nature*, vol. 588, pp. 604–609, Dec. 2020. Publisher: Nature Publishing Group.

[7] E. Puiutta and E. M. Veith, "Explainable reinforcement learning: A survey," 2020.

[8] G. A. Vouros, "Explainable Deep Reinforcement Learning: State of the Art and Challenges," *ACM Comput. Surv.*, vol. 55, pp. 92:1–92:39, Dec. 2022.

[9] T. Hickling, A. Zenati, N. Aouf, and P. Spencer, "Explainability in Deep Reinforcement Learning: A Review into Current Methods and Applications," *ACM Comput. Surv.*, vol. 56, pp. 125:1–125:35, Dec. 2023.

[10] M. Mersha, K. Lam, J. Wood, A. K. AlShami, and J. Kalita, "Explainable artificial intelligence: A survey of needs, techniques, applications, and future direction," *Neurocomputing*, vol. 599, Sept. 2024.

[11] L. Longo, M. Brcic, F. Cabitza, J. Choi, R. Confalonieri, J. D. Ser, R. Guidotti, Y. Hayashi, F. Herrera, A. Holzinger, R. Jiang, H. Khosravi, F. Lecue, G. Malgieri, A. PÃ¡ez, W. Samek, J. Schneider, T. Speith, and S. Stumpf, "Explainable Artificial Intelligence (XAI) 2.0: A manifesto of open challenges and interdisciplinary research directions," *Information Fusion*, vol. 106, June 2024.

[12] P. KÖRNER, M. LEUSCHEL, J. BARBOSA, V. S. COSTA, V. DAHL, M. V. HERMENEGILDO, J. F. MORALES, J. WIELEMAKER, D. DIAZ, S. ABREU, and et al., "Fifty years of prolog and beyond," *Theory and Practice of Logic Programming*, vol. 22, no. 6, p. 776–858, 2022.

[13] H. J. Levesque, R. Reiter, Y. Lespérance, F. Lin, and R. B. Scherl, "Golog: A logic programming language for dynamic domains," *The Journal of Logic Programming*, vol. 31, no. 1, pp. 59–83, 1997. Reasoning about Action and Change.

[14] A. Ferrein and G. Lakemeyer, "Logic-based robot control in highly dynamic domains," *Robot. Auton. Syst.*, vol. 56, p. 980–991, Nov. 2008.

[15] B. Errico and L. C. Aiello, "Intelligent agents in the situation calculus: An application to user modelling," in *Practical Reasoning* (D. M. Gabbay and H. J. Ohlbach, eds.), (Berlin, Heidelberg), pp. 126–140, Springer Berlin Heidelberg, 1996.

[16] T. Hofmann, T. Niemueller, J. Claßen, and G. Lakemeyer, "Continual planning in golog," in *Proceedings of the Thirtieth AAAI Conference on Artificial Intelligence*, AAAI'16, p. 3346–3353, AAAI Press, 2016.

[17] S. Mehdipour Ataee and View Profile, "A frame and first-order logic solution for the Wumpus World," *Expert Systems with Applications: An International Journal*, vol. 220, June 2023. Publisher: Pergamon Press, Inc.

[18] R. R. Selvaraju, M. Cogswell, A. Das, R. Vedantam, D. Parikh, and D. Batra, "Grad-CAM: Visual Explanations from Deep Networks via Gradient-Based Localization," in *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 618–626, Oct. 2017. ISSN: 2380-7504.

[19] M. T. Ribeiro, S. Singh, and C. Guestrin, ""Why Should I Trust You?": Explaining the Predictions of Any Classifier," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, (New York, NY, USA), pp. 1135–1144, Association for Computing Machinery, Aug. 2016.

[20] S. M. Lundberg and S.-I. Lee, "A unified approach to interpreting model predictions," in *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, (Red Hook, NY, USA), pp. 4768–4777, Curran Associates Inc., Dec. 2017.

[21] T. Bewley, J. Lawry, and A. Richards, "Summarising and comparing agent dynamics with contrastive spatiotemporal abstraction," in *XAI-IJCAI22 Workshop*, 2022.

[22] Y. Bekkemoen and H. Langseth, "ASAP: Attention-Based State Space Abstraction for Policy Summarization," in *Proceedings of the 15th Asian Conference on Machine Learning*, pp. 137–152, PMLR, Feb. 2024. ISSN: 2640-3498.

[23] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Explainable reinforcement learning through a causal lens," in *Proceedings of the Thirty-Fourth AAAI Conference on Artificial Intelligence*, 2020.

[24] P. Madumal, T. Miller, L. Sonenberg, and F. Vetere, "Distal explanations for explainable reinforcement learning agents," *arXiv preprint arXiv:2001.10284*, vol. 2, 2020.

[25] W. Shi, G. Huang, S. Song, Z. Wang, T. Lin, and C. Wu, "Self-supervised discovering of interpretable features for reinforcement learning," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 5, pp. 2712–2724, 2020.

[26] D. Hein, S. Udluft, and T. A. Runkler, "Interpretable policies for reinforcement learning by genetic programming," 2018.

[27] A. Verma, V. Murali, R. Singh, P. Kohli, and S. Chaudhuri, "Programmatically interpretable reinforcement learning," 2019.

[28] C. Geng, X. Xu, A. Xing, Z. Zhao, and X. Si, "From neural representations to interpretable logic rules," 2025.

[29] C. Persia and A. Ozaki, "Extracting rules from neural networks with partial interpretations," *Proceedings of the Northern Lights Deep Learning Workshop*, vol. 3, Mar. 2022.

[30] M. Craven and J. Shavlik, "Extracting Tree-Structured Representations of Trained Networks," in *Advances in Neural Information Processing Systems*, vol. 8, MIT Press, 1995.

[31] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*. USA: Prentice Hall Press, 3rd ed., 2009.

[32] S. Ahlawat, "Wumpus World." `https://github.com/swapnil-ahlawat/Wumpus-World`, 2020. Accessed: 2025-07-24.

[33] K. KC, "Wumpus World Prolog." `https://github.com/kckusal/wumpus-world`, 2019. Accessed: 2025-07-24.

[34] J. Szymanski, J. Dobrosolski, H. Mora, and K. Draszawka, "Neural network agents trained by declarative programming tutors," in *2024 IEEE Congress on Evolutionary Computation (CEC)*, pp. 1–7, 2024.

[35] N. Kumar, "Wumpus RL." `https://github.com/nowke/wumpus-rl`, 2019. Accessed: 2025-07-24.

[36] K. Draszawka, J. Szymański, D. Gil, M. T. S. Pont, and H. Mora, "Learning action strategies in the wumpus world with dqn," in *Proceedings of the 17th International Conference on Computational Collective Intelligence (ICCCI 2025)*, 2025. To appear.

[37] G. Liu, O. Schulte, W. Zhu, and Q. Li, "Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees," in *Machine Learning and Knowledge Discovery in Databases: European Conference, ECML PKDD 2018, Dublin, Ireland, September 10â€“14, 2018, Proceedings, Part II*, (Berlin, Heidelberg), pp. 414–429, Springer-Verlag, Sept. 2018.

[38] O. Bastani, Y. Pu, and A. Solar-Lezama, "Verifiable reinforcement learning via policy extraction," in *Proceedings of the 32nd International Conference on Neural Information Processing Systems*, NIPS'18, (Red Hook, NY, USA), pp. 2499–2509, Curran Associates Inc., Dec. 2018.

[39] J. Hancock, *Jaccard Distance (Jaccard Index, Jaccard Similarity Coefficient)*. 10 2004.

[40] A. F. Agarap, "Deep learning using rectified linear units (relu)," 2019.

[41] R. Girshick, "Fast r-cnn," in *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV '15, (USA), p. 1440–1448, IEEE Computer Society, 2015.

[42] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017.