

Extracting Tactics Learned from Self-Play in General Games

Dennis J. N. J. Soemers^{a,*}, Spyridon Samothrakis^b, Éric Piette^a, Matthew Stephenson^a

^a*Maastricht University, Department of Advanced Computing Sciences, Paul-Henri Spaaklaan 1, 6229 EN, Maastricht, the Netherlands*

^b*University of Essex, Institute for Analytics and Data Science, Wivenhoe Park, Colchester CO4 3SQ, UK*

Abstract

Local, spatial state-action features can be used to effectively train linear policies from self-play in a wide variety of board games. Such policies can play games directly, or be used to bias tree search agents. However, the resulting feature sets can be large, with a significant amount of overlap and redundancies between features. This is a problem for two reasons. Firstly, large feature sets can be computationally expensive, which reduces the playing strength of agents based on them. Secondly, redundancies and correlations between features impair the ability for humans to analyse, interpret or understand the tactics learned by the policies. We look towards decision trees for their ability to perform feature selection, and serve as interpretable models. Previous work on distilling policies into decision trees uses states as inputs, and distributions over the complete action space as outputs. In contrast, we propose and evaluate a variety of decision tree types, which take state-action pairs as inputs, and provide various different types of outputs on a per-action basis. An empirical evaluation over forty-three different board games is presented, and two of those games are used as case studies where we take a more detailed look and attempt to interpret the discovered features.

Keywords: games, feature selection, decision trees, explainable AI

*Corresponding author

Email address: `dennis.soemers@maastrichtuniversity.nl` (Dennis J. N. J. Soemers)

1. Introduction

Machine learning techniques have been shown to be capable of producing strong or even superhuman game-playing agents for a variety of games, but identifying just the key, basic tactics in an explicit *white-box* format for general
5 games remains a challenge. State-of-the-art results in terms of game-playing strength [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] are, in recent years, essentially always obtained using Deep Neural Networks (DNNs) [11] as function approximators for functions such as policies and/or value functions [12]. Such approaches based on deep learning require significant computational resources [13, 14] even for just a
10 single game, which makes their use prohibitive for research projects that require scaling up to orders of 1,000 or more different games [15, 16]. Additionally, despite numerous efforts, interpretability of DNNs remains a concern [17, 18]. Explaining game playing agents is crucial if one is interested in using these agents as teaching aids, rather than black-box adversaries.

15 As a less computationally intensive alternative, simple linear functions of state-action features have been proposed [19, 20]. Figure 1 provides some intuition for what such state-action features look like and what they may encode. This approach generally does not lead to state-of-the-art or superhuman playing strength, but can allow for meaningful policies to be trained in a wide variety
20 of games in relatively short amounts of time—for example, by using only 100 or 200 games of self-play [21, 22], in contrast to the many millions typically used for deep learning.

The set of features discovered and used by these training processes often contain many strongly correlated or otherwise redundant features, leading to
25 two issues. Firstly, this can negatively affect the playing strength of the resulting agents [21], because greater numbers of features are associated with greater computational costs, which reduces the space explored by search algorithms such as Monte-Carlo tree search (MCTS) [23, 24, 25]. Secondly, strong correlations between features, and presence of redundant features, can hamper explainabil-

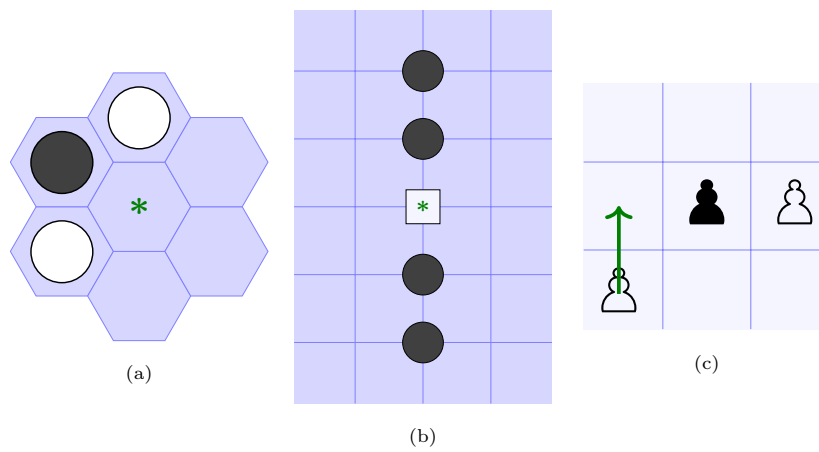


Figure 1: Three examples of local state-action features that may be useful in various games. A small square indicates a position that must be empty. Uncovered sites are drawn for ease of interpretation, but play no role in the feature; they may be empty or non-empty or even not exist at all. (a) This feature matches actions that either complete or break a “bridge” of white pieces, depending on which player is the player to move. (b) This feature matches actions that either complete or break a line of five black pieces. (c) This feature matches actions that move the bottom-left white pawn in such a way that the black pawn becomes flanked by two white pawns.

ity and interpretability. This is the case in particular when directly inspecting the trained weights of a linear model without accounting for such correlations [26, 27], but also when using more advanced methods, such as permute-and-predict methods, for estimating feature importance [28]. Because our set of features grows dynamically during the training process (as discussed in Subsection 2.4), and training from self-play inherently leads to a non-stationary data stream, correlations and dependences between features can change over time. Some features may not be redundant initially, but become redundant later on in a training process, or vice versa. This reduces the likelihood that on-line approaches such as ℓ_1 regularisation [29] can effectively identify and ignore redundant features.

This paper explores how to address these two issues and extract meaningful, understandable, and explainable tactics after training such linear models via self-play, by distilling them into a variety of decision or regression trees. Such trees will ideally select a relatively small number of key features to represent relevant tactics for any game under consideration. The primary focus of this paper is on proposing and evaluating several different ways in which the outputs of decision trees can be represented, since we find that existing approaches provide limited support for domains with large and variable action spaces. Note that, in terms of explainability, the primary interest is in providing useful but basic tactics for beginners, which could, for instance, be included in automatically-generated manuals for new (possibly procedurally-generated) or otherwise unknown games [30]. Automatically generating insights into basic tactics for games may also be interesting to aid game designers, and can also be used to gain a deeper, more explicit understanding of what our algorithms manage to learn and in which situations they fail to learn.

The remainder of this paper is structured as follows. Section 2 provides background information on work that this paper builds on, as well as other related work. A discussion of various ways in which different features can have strong correlations or other dependences is provided in Section 3. Several different types of decision trees, with state-action features as inputs and various

different output representations, are proposed and discussed in Section 4. Section 5 describes an evaluation of the proposed techniques, and Section 6 finally concludes the paper.

2. Background

65 This section discusses related work on explainability in reinforcement learning in games, and it provides background information on the considered problem setting and the training processes used to generate the initial sets of features and policies. These policies are the ones that are subsequently distilled into smaller, more interpretable policies using smaller sets of features in the remainder of the
70 paper.

2.1. Related Work on Explainability in Reinforcement Learning and Games

In related work on explainability in games, and reinforcement learning (RL) more generally, there is often a focus on (1) *local* explanations, which are explanations on a per-state basis, (2) explaining value functions, and/or (3) explaining
75 policies in domains with fixed, and relatively small, avatar-centric action spaces. For example, Lin et al. [31] focus on generating contrastive explanations for a model’s preference of one action over another for specific states. Baier and Kaisers [32, 33] consider the problem of explaining individual decisions made by tree search algorithms such as MCTS, and Silva et al. [34] generate counterfac-
80 tual justifications for decisions made by an adversarial tree search for Curling. Pálsson and Björnsson [35] generate visualisations of the most important parts of the state representation for the predictions made by a value function on a per-state basis in the game of Breakthrough, and Hilton et al. [36] similarly visualise important parts of the state for policies and value functions in the
85 CoinRun environment. In contrast, the aim in this paper is to extract general, simple tactics that can be explained to humans for general use throughout an entire game (or substantial portions thereof).

Coppens et al. [37, 38] and Deproost [39] distill trained policies into various forms of decision trees and rules, which can lead to local (state-specific)

90 as well as global (game-wide) explanations of policies. These were evaluated
in environments such as the Mario AI benchmark, Ms Pacman, and Enduro.
Other commonly-used environments in work on explainable RL are CoinRun
[36], Lunar Lander, Cart Pole, and Mountain Car [40, 31]. These are all envi-
ronments with fixed and relatively small action spaces, where actions can easily
95 be labelled and understood when used as outputs for a classifier. These are
often actions such as “left”, “right”, “jump”, and so on, which are typically used
to control a single avatar. In contrast, this paper considers (board) games with
significantly larger action spaces, where the subsets of the action space that are
legal can also vary from state to state. Similar approaches, where all possibly
100 unique actions are enumerated as potential target classes for a classifier, quickly
lead to decision trees or rules that become difficult to understand when they
have to distinguish between hundreds or thousands or even more distinct target
classes.

McGrath et al. [41] describe an extensive analysis of the state-of-the-art
105 Chess engine of AlphaZero, attempting to gain insight into which human-understandable
concepts it learns, and when it does so throughout its training process. The ma-
jority of this analysis assumes that deep, expert human knowledge is already
explicitly available (to actively probe the network during training for such con-
cepts), and relies on massive amounts of self-play data for a single game. Neither
110 of these are assumed to be available in this paper. Interestingly, their analysis
suggests that AlphaZero tends to learn tactical skills before it learns positional
skills. This may be an artifact of how the training process (based on the use
of tree search in self-play) works, or may be an indication that learning local
tactics is inherently easier than, or a prerequisite for, learning global strategies.
115 This intuition is one of the reasons that, given the assumed computational con-
straints and requirements for extremely short training runs in this paper, the
focus is placed on learning policies based on local patterns, rather than functions
such as state-value functions, which operate on a more global level.

2.2. Markov Decision Processes

120 This paper uses the standard formalism of Markov decision processes (MDPs),
as commonly used in RL [12] to formalise the problem setting. An MDP is
defined by a set of states \mathcal{S} , a set of actions \mathcal{A} , an initial state $s_0 \in \mathcal{S}$, and dynamics
 \mathcal{P} such that $\mathcal{P}(s', r | s, a)$ denotes the probability of transitioning into a succes-
sor state $s' \in \mathcal{S}$ and obtaining a real-valued reward r , when selecting an action
125 $a \in \mathcal{A}$ from a current state $s \in \mathcal{S}$. The set of legal actions may be restricted
depending on the current state, and $\mathcal{A}(s) \subseteq \mathcal{A}$ is used to denote the actions
that are legal in a state s . The behaviour of an agent is typically described as
a policy π , such that $0 \leq \pi(s, a) \leq 1$ denotes the probability that the agent
selects an action $a \in \mathcal{A}$ when it is in a state s , and $\sum_{a \in \mathcal{A}(s)} \pi(s, a) = 1$.

130 Note that the standard MDP formalism applies to a single agent, but the
games considered in this paper are actually environments with multiple (typi-
cally 2) agents, who often have opposing objectives. This is important to take
into account when designing self-play training algorithms to train policies π ,
but in this paper it is assumed that such policies have already been trained [21].
135 Given this assumption, it is safe to use the standard MDP formalism throughout
this paper, implicitly assuming that any influence of other agents has already
been absorbed into the dynamics \mathcal{P} .

2.3. Spatial State-Action Features

In previous work [19, 20], we proposed a formalisation for *spatial state-action*
140 *features* that allows for applicability to a wide variety (board) games—essentially
any game that involves 2-dimensional, discrete areas, with spatial semantics
having some degree of relevance to game-play, is supported. For any given
state-action pair (s, a) , where s denotes a game state and a an action that is
legal in s , such a feature tests whether a certain pattern (or configuration) of
145 requirements in the local area around the action a matches in the state s . For
example, a feature can test whether the destination of a move is next to a
friendly or enemy piece, whether it moves away from a position next to the edge
of the board, or any other combination of one or more such conditions for one

or more positions specified relative to positions affected by a . Several examples
150 are depicted in Figure 1.

Similar types of local patterns have also been used in several game-specific
engines, for games such as Breakthrough [42], Chess [43], Go [44, 45, 46, 47, 48],
and Hex [49, 50]. The work described by Soemers et al. [20] focuses on formal-
ising such patterns in a way that is compatible with many different games—
155 irrespective of the particular board geometry used [51]—and an efficient imple-
mentation for pattern matching without game-specific domain knowledge in the
Ludii general game system [52, 53].

2.4. Feature Discovery and Policy Training

In previous work on the use of patterns in games such as Go, it is relatively
160 common to exhaustively enumerate all patterns of a given size (e.g., all 3×3
patterns centered on the intersection under consideration) [54, 55, 48, 56, 57].
When considering games with arbitrarily complex board geometries [51] or sig-
nificantly greater numbers of types of distinct pieces than in Go (e.g., twelve
in Chess versus two in Go), it is no longer feasible to exhaustively generate all
165 such patterns even for a small size.

For this reason, an approach is used where discovery of new features and
training of policies using those features are intertwined in a self-play training
process [21]. This starts with a smaller set of simple *atomic* features, and new
features are iteratively constructed by combining existing features (or rotated
170 or reflected instances of them) into more complex compound patterns [58, 59,
60, 21]. This process is depicted in Figure 2. The atomic features that the
process starts with are features that only have a single requirement for the game
state data, in addition to any requirements they may have for action data. For
example, an atomic feature may require a single site (relative to some reference
175 point) to be occupied by a white stone.

Given a (dynamically growing) set of features, a parameterised policy π_{θ} is
trained from self-play by learning a vector of parameters $\theta = [\theta_0, \theta_1, \dots, \theta_{n-1}]$.
Such a vector contains one parameter (or weight) θ_i for every feature ϕ_i in a set

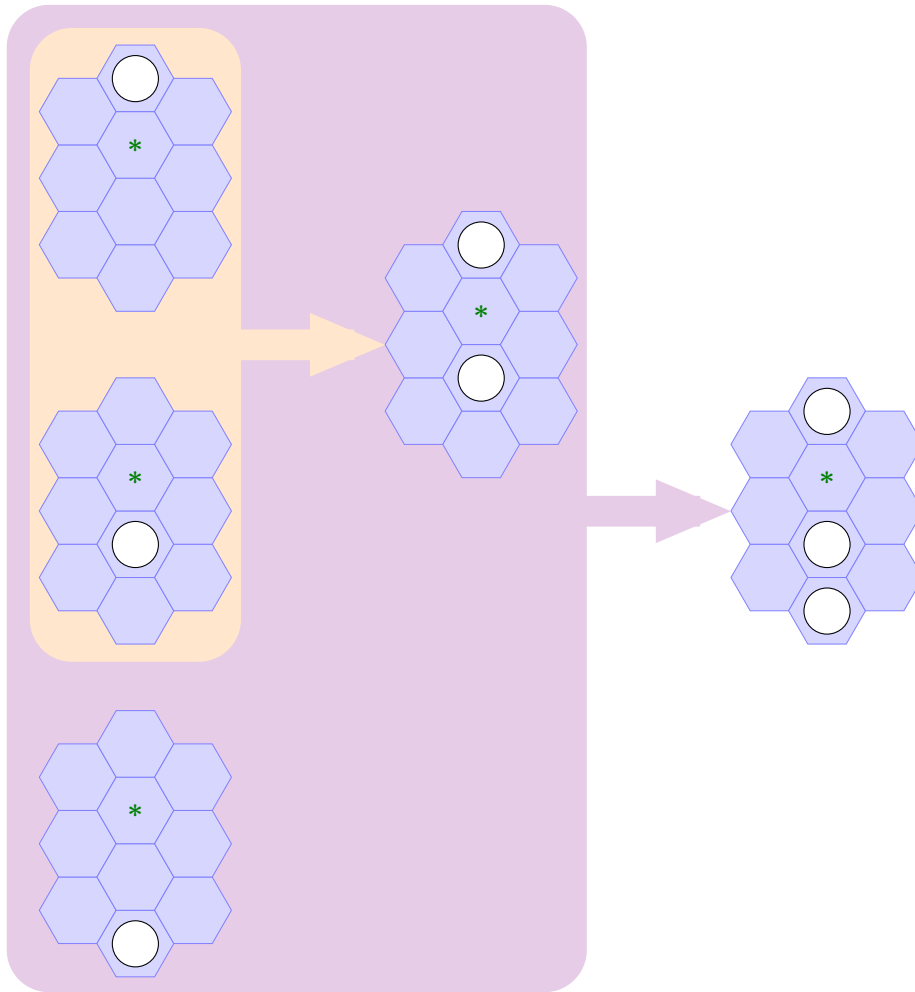


Figure 2: Example of how new features are generated by combining instances of existing features. On the left-hand side, we start with three instances of atomic features. The top and middle instances are first combined into a new feature that matches actions that place a stone in between two white stones. This more complex feature is subsequently combined again with another feature, finally resulting in a feature that matches actions that complete or break a line of four white stones.

of n features. Whenever new features are discovered and added to the set during a training process, new parameters—initialised to a value of 0—are appended to the parameter vector. Features $\phi_i : \mathcal{S} \times \mathcal{A}(s) \rightarrow \{0, 1\}$ are binary features that take values of either $\phi_i(s, a) = 0$ or $\phi_i(s, a) = 1$ for any input state-action pair (s, a) . A boldface $\boldsymbol{\phi}(s, a) = [\phi_0(s, a), \phi_1(s, a), \dots, \phi_{n-1}(s, a)]$ is used to denote a vector of such feature values for a state-action pair (s, a) . The dot product between a feature vector and a trained parameter vector produces a *logit* $z_{\boldsymbol{\theta}}(s, a) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)$. For any given state s , the probabilities $\pi_{\boldsymbol{\theta}}(s, a)$ of all the legal actions $a \in \mathcal{A}(s)$ of the policy are then computed by a softmax over the logits, as in Equation 1:

$$\pi_{\boldsymbol{\theta}}(s, a) = \frac{\exp(z_{\boldsymbol{\theta}}(s, a))}{\sum_{a' \in \mathcal{A}(s)} \exp(z_{\boldsymbol{\theta}}(s, a'))} \quad (1)$$

The policies considered in this paper—which are to be distilled into decision trees—were trained in a similar way as the policies in AlphaZero [5], which means that they were trained using a cross-entropy loss to mimic the behaviour of a search-based agent. This may be viewed as a form of multinomial logistic regression, albeit with a non-stationary target distribution, the performance of which is meant to improve in terms of playing strength as training progresses. For further details on the setup of the self-play training processes used to train initial policies for the experiments in this paper, we refer to our earlier work [20].

3. Feature Dependences

Sets of features constructed and used as described in Subsection 2.4 frequently contain many subsets of features with strong correlations or dependences between each other. This can be considered problematic for two reasons. Firstly, if there are many redundancies in the set of features, computing policies may be slower than necessary, which harms playing strength of tree search algorithms guided by such a policy [21]. Secondly, understanding, interpreting, or analysing the importance of features based on their trained weights becomes

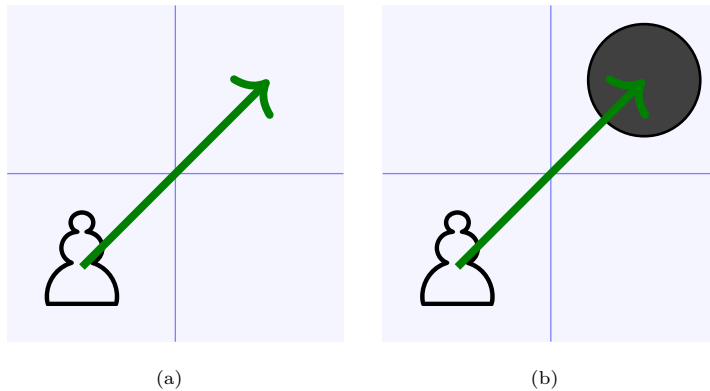


Figure 3: Two example features that may or may not be equivalent depending on the game they are used in. Both encode, for grids of square cells, a diagonal move by a white pawn, but (b) has the additional restriction that the destination must be occupied by a black piece (of any type, indicated by a circle).

error-prone when features are not mutually independent [26, 27, 28]. In this section, several different ways in which (strong) dependences between features
 195 may exist are distinguished.

3.1. Game-Agnostic Dependences

For some pairs of features ϕ_i and ϕ_j , $i \neq j$, we have that one of them being (in)active by definition—regardless of which game is being played—implies the other also being (in)active, i.e. $(\phi_i(s, a) = 0) \Rightarrow (\phi_j(s, a) = 0)$ or $(\phi_i(s, a) = 1) \Rightarrow (\phi_j(s, a) = 1)$. Consider, for example, the different features depicted in
 200 Figure 2. By definition, whenever a feature that was constructed by combining a pair of other features is active, its constituents must also be active. Conversely, whenever a simpler feature is not active, any compound feature with the simpler feature as a constituent also cannot be active. Note that these are just examples:
 205 there may also be similar implications between features that are not each other’s constituents. For example, a feature that requires at least one friendly adjacent piece being active will always imply that a feature requiring at least one non-empty adjacent position is also active.

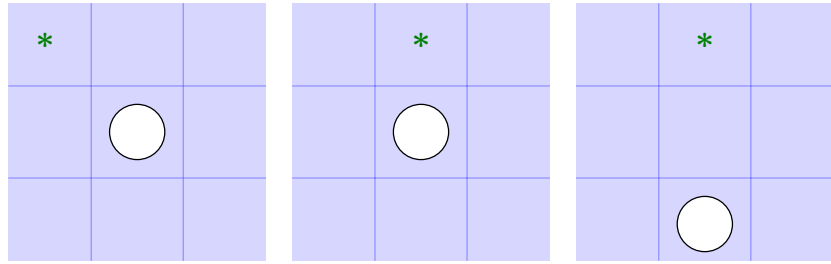
3.2. Emergent Dependences from Game Rules

210 Some pairs of features ϕ_i and ϕ_j , $i \neq j$, may have strong correlations or implications between each other only in certain games, as a result of such a game’s rules. For example, in the game of Chess, pawns are only allowed to move diagonally if that results in a capture of an opposing piece. Hence, in this game, a feature that matches diagonal pawn moves correlates perfectly with a
215 feature that matches diagonal pawn moves towards an enemy piece. This is depicted in Figure 3. Different games may also allow pawns to move diagonally towards empty positions, and in such games these two features would no longer be equivalent. Breakthrough is an example of such a game. These features would still have a strong dependence as described in the previous subsection,
220 but may not correlate perfectly in all games.

A related issue is that some features have extremely low or high marginal probabilities of being active, and are therefore uninformative, as a result of a game’s rules. For example, in the game of Tic-Tac-Toe, features that require the destination of an action to be empty are always active because this is also
225 a requirement for any move to be legal in this game. Similarly, a feature that requires the destination of an action to be within two steps of an edge of the board is always active because this game is played on a grid of 3×3 cells.

3.3. Emergent Dependences from Policies

Finally, there can be dependences between features that are not necessarily
230 due to the particular game being played, but rather due to the policies used to generate playing experience in them. By design, the self-play approaches used to generate experience for feature discovery and weight training [2, 21] have some degree of exploration—for diversity in generated experience—but also a clear bias towards selecting actions that are considered to be strong by the agent
235 used in self-play. This causes the distribution of states that are experienced—and therefore also legal actions that are observed—to be highly non-uniform. This can result in certain pairs of features very frequently or very rarely co-occurring *in practice*, even if perhaps they would not when observing gameplay



(a) This feature matches actions diagonally adjacent to a white stone.
 (b) This feature matches actions orthogonally adjacent to a white stone.
 (c) This feature matches actions at two orthogonal steps from a white stone.

Figure 4: MCTS-based players have a strong tendency to open games of Tic-Tac-Toe in the centre of the board. Therefore, it is highly likely for every action available in the first turn of the second player to match either the feature depicted in (a) or the one in (b). If the first player opens in a corner, the second player also has legal moves in their first turn that match the feature depicted in (c), but this is rarely observed in self-play between MCTS agents.

from different agents.

240 Consider, for example, the game of Tic-Tac-Toe, in which players take turns placing pieces on a 3×3 grid, and the first player to complete a line of three wins. MCTS-based players almost always open the game by playing in the centre of the board.¹ This causes certain features (see Figure 4) to have very high or low marginal probabilities of matching any legal actions in the first turn
 245 of the second player, which could be different if other openings were observed more frequently. These high or low marginal probabilities also lead to high or low co-occurrences with other features that may be less affected by the different openings.

3.4. Discussion

250 The subsections above describe various types of dependences between features. Each of these can lead to situations where the probabilities of being

¹Players based on algorithms such as $\alpha\beta$ -search are more likely to also open in one of the corners, but MCTS tends to have a preference for the centre of the board because this has a greater probability of leading to wins against random players.

(in)active for some features can be predicted with high (sometimes perfect) accuracy based on the activity of other features. This can lead to redundancies in sets of features, where some are “unnecessary” in the presence of others. In terms
255 of computational costs, this is not a major issue in the case of game-agnostic dependences (Subsection 3.1), because pattern matching is performed using the highly efficient “SPatterNet” approach [20], which already leverages such relations to speed up pattern matching. More specifically, this is a technique that aims to optimise the order in which propositions are evaluated for pattern
260 matching with a larger set of features, and it can automatically account for the game-agnostic relations in this process as listed in Table 1. However, the other types of dependences cannot be accounted for without domain knowledge of the particular game being played or the agents that are playing, which means that redundancies due to these other types are harmful in terms of computational
265 efficiency.

In terms of understanding or explaining policies, it is also important to keep these dependences in mind. The single weight of an individual feature, without accounting for features that are likely to co-occur or likely *not* to co-occur, and their weights, does not necessarily give a good idea of the strength
270 of actions for which that feature matches. Furthermore, especially in the case of perfect correlations, there can be different features that provide equally valid explanations in theory, but where some may be subjectively viewed as more representative than others. For example, the two features depicted in Figure 3 could form equally valid explanations for the idea that using a pawn to capture
275 an enemy is a strong (or weak) move in Chess, but we imagine that humans may prefer the rightmost feature since it more explicitly also visualises the enemy piece.

4. Decision Trees of State-Action Features

Since decision trees are generally considered to be inherently interpretable
280 models [27, 17], and can also be used to select the most important features

Proposition a	Propositions proven by a
x is empty	x is empty x is not owned by p (for any $p > 0$) x is not piece i (for any $i > 0$)
x is not empty	x is not empty
x is owned by p	x is owned by p x is not piece i (for any i not owned by p) x is piece i (if i is the sole type owned by p) x is not empty
x is not owned by p	x is not owned by p x is not piece i (for any i owned by p)
x is piece i	x is piece i x is not empty x is not piece j (for any $j \neq i$) x is owned by p (where p is the owner of i) x is not owned by p (for any p that does not own i)
x is not piece i	x is not piece i x is not owned by p (if i is the sole type owned by p)

Table 1: Game-agnostic relations between propositions in features that the SPatterNet approach [20] for pattern matching can automatically account for. Propositions a in the left column, when true, always imply the matching propositions in the right column. In every proposition, x denotes a site (i.e., a cell or an intersection of a game board). Redundancies in feature sets due to these relations are therefore not harmful in terms of computational efficiency. Table reproduced from [20].

[61], we look towards them for feature selection (with the ultimate goal of reducing computational overhead whilst preserving strong policies) as well as the extraction of explainable tactics.

4.1. Decision Tree Structures

285 When using function approximators for policy training in games or RL, it is customary for these functions (often neural networks) to take a representation of a state s as input, and produce one logit $z(s, a)$ as output for every action a that may possibly be legal in *any* state s [12]. Resulting vectors of logits are transformed into discrete probability distributions over the actions by applying
290 a softmax (plus invalid action masking [62] in games where some actions are sometimes illegal). When such policies are subsequently distilled into decision trees or rules for explainability, these are typically trained as classifiers that again use representations of states s as input (splitting on features of states), and produce probability distributions over all actions (which must all be explicitly
295 enumerated as potential target classes) as outputs [37, 38, 39]. An example of what such a decision tree could look like is depicted in Figure 5. In the example case of Tic-Tac-Toe, every leaf node outputs a probability distribution over nine possible actions (some of which may be illegal depending on the input game state s). In more complex games such as Chess or Shogi, the output distributions
300 would have to be defined for thousands of different elements [5].

The input and output structures for the original (linear) policies considered in this paper are different, since they take a representation of a state-action pair (s, a) as input, and produce only a single logit for that same (s, a) pair as output. Hence, the most straightforward way to distill such a policy into
305 a decision tree would be to build a *regression* tree that takes representations of state-action pairs as input (splitting on state-action features), and produces individual logits as outputs. An example of such a tree is depicted in Figure 6. In the remainder of this subsection, advantages and disadvantages of this approach, as well as several other possibilities for structuring (in particular the outputs
310 of) decision trees, are discussed.

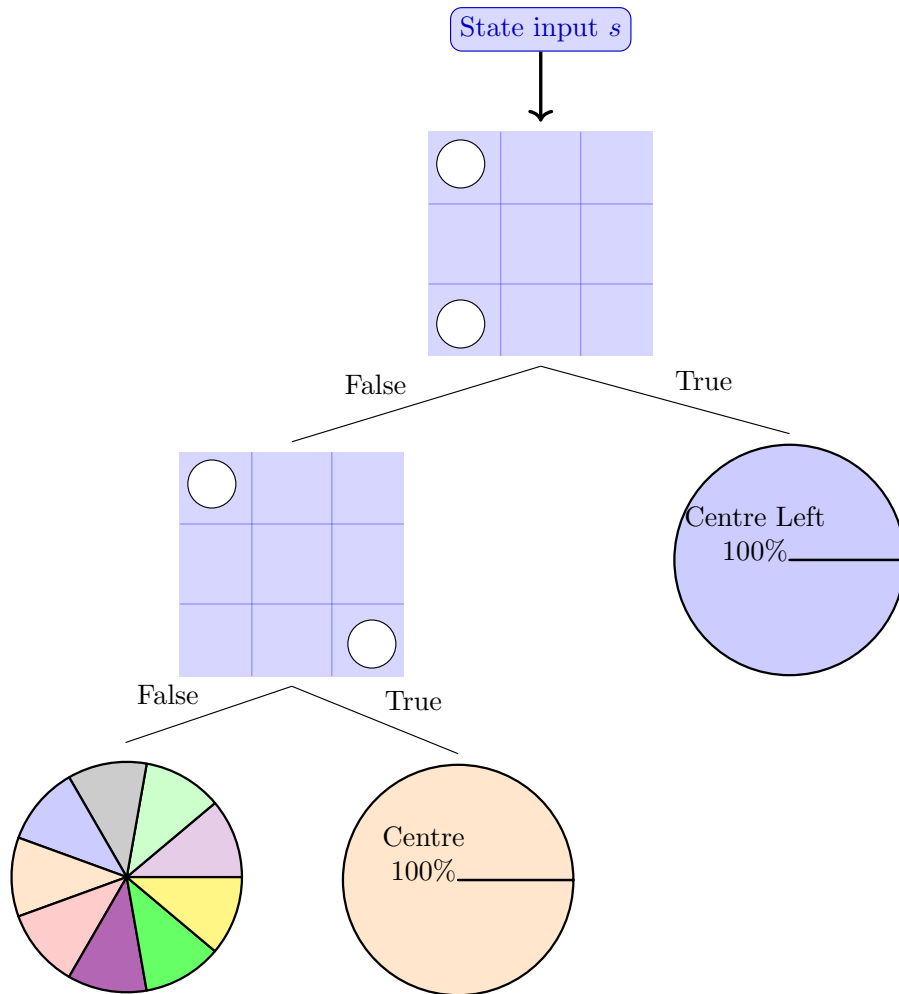


Figure 5: Example of a handcrafted policy for Tic-Tac-Toe, modelled as decision tree that takes states as inputs and produces probability distributions over all possible actions as outputs. This decision tree can recognise two particular cases of winning actions for the white player, but otherwise recommends a uniform distribution over all actions.

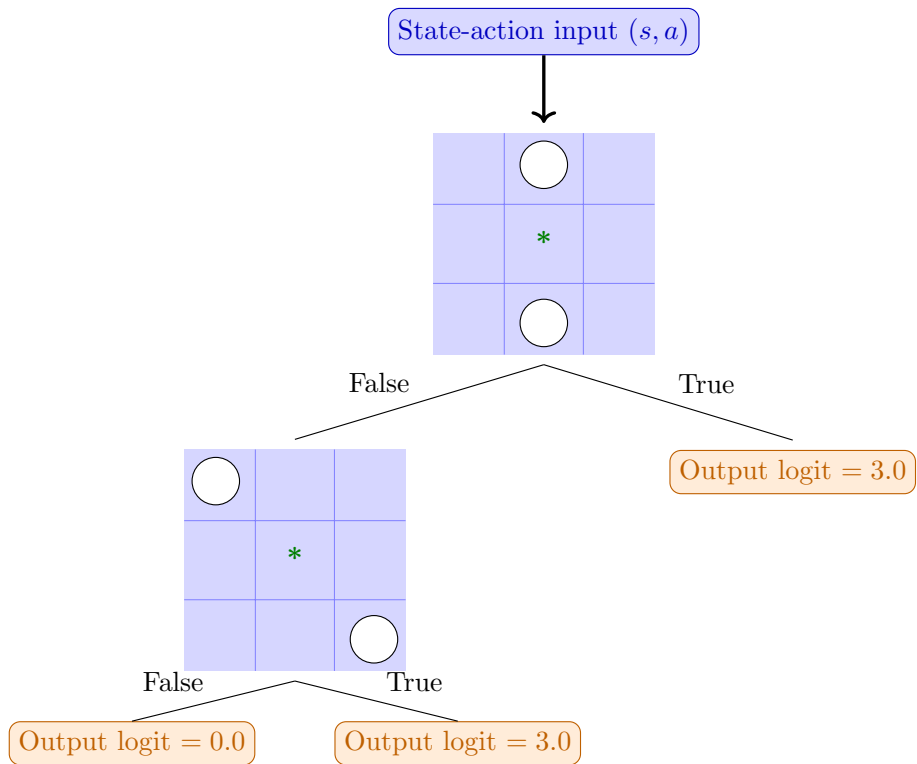


Figure 6: Example logit regression tree for a handcrafted policy for Tic-Tac-Toe, modelled as a regression tree that takes state-action pairs (s, a) as inputs and produces a single logit for such a pair as output. This regression tree can detect actions that complete any (assuming local rotations and reflections of features are used, which we do) orthogonal or diagonal line by placing the third stone in the middle of such a line, and assigns logit values of 3.0 to such actions. Any other action is assigned a logit value of 0.0. These outputs are meaningless on their own, but in combination with logits for other legal actions can easily be transformed into probabilities by a computer program.

4.1.1. Logit Regression Trees

In terms of raw playing strength, regression trees that output logits—exactly as our original policies do—may be expected to have the highest potential performance. Such a *logit regression tree* is at least as expressive as a linear policy is, and—in contrast to some of the other structures described below—does not involve any additional approximations. In fact, such a tree could even be *more* expressive than a linear policy, because decision trees are non-linear functions of their input features.

In terms of explainability, we argue that single-logit outputs can be problematic. Every individual feature used in branching points, as well as the entire path from root to leaf node, could be considered interpretable, but the logit output itself would be difficult to understand. In isolation, a logit value $z(s, a)$ does not have any meaning. A logit $z(s, a)$ only gains some meaning when it is compared to another logit $z(s, a')$ for a different action $a' \neq a$ that is legal in the same state s , and even then the exact relationship is somewhat difficult to understand. The exact relationship is that the ratio of probabilities assigned to two actions by a policy π is given by the ratio of the exponentials of their logits:

$$\frac{\pi(s, a)}{\pi(s, a')} = \frac{\exp(z(s, a))}{\sum_{b \in \mathcal{A}(s)} \exp(z(s, b))} \times \frac{\sum_{b \in \mathcal{A}(s)} \exp(z(s, b))}{\exp(z(s, a'))} = \frac{\exp(z(s, a))}{\exp(z(s, a'))} \quad (2)$$

This relationship is arguably not nearly as easy to interpret as the direct probabilities assigned to *all* actions at once by classifier trees that take states as inputs and enumerate all actions (in domains with small and fixed action spaces) as potential target classes.

4.1.2. Multiclass State-Action Classification Trees

While the logit outputs $z(s, a)$ for state-action pairs (s, a) discussed above can be informative for a software agent, they may be difficult for humans to interpret. If the goal is to help humans easily recognise actions that are likely to be weak or strong in general based on local patterns around such actions, it may be more helpful for a decision tree to be trained to explicitly provide

outputs that can be directly interpreted as such qualitative estimates of action
330 quality. Following this intuition, we propose to train a decision tree that takes
state-action pairs (s, a) as input, and as output classifies that action in that state
as belonging to one out of a small selection of classes, each of which provides
a qualitative judgement of action quality. More specifically, the following three
classes are used in the implementation and experiments discussed in this paper,
335 but different partitions would also be possible:

1. *Bottom 25%*: label assigned to actions a that are predicted to be among
the worst 25% of legal actions $\mathcal{A}(s)$ in the state s .
2. *IQR*: label assigned to actions a that are predicted to be in the interquar-
tile range (better than bottom 25%, but worse than top 25%) of legal
340 actions $\mathcal{A}(s)$ in the state s .
3. *Top 25%*: label assigned to actions a that are predicted to be among the
best 25% of legal actions $\mathcal{A}(s)$ in the state s .

Figure 7 depicts an example of such a tree.

In comparison to logit regression trees, multiclass state-action classification
345 trees involve an additional level of approximation in the sense that larger col-
lections of inputs that would have distinct outputs in a logit regression tree are
grouped together and are assigned identical target labels for this type of classi-
fication tree. This may be expected to lead to a lower level of playing strength
when used to control a policy, but the output representation is arguably easier
350 to interpret.

4.1.3. Best-Action Classification Trees

One potential issue with multiclass state-action classification trees as de-
scribed above is that there is a natural ordering of the output classes (i.e., *Top*
25% > *IQR* > *Bottom 25%*), but neither the model nor the decision tree in-
355 duction algorithm account for this in any way. For example, it is possible for
a tree to predict equal probabilities of 0.5 for the *Bottom 25%* and *Top 25%*
classes for a given input pair (s, a) , with a probability of 0.0 for the *IQR* class.

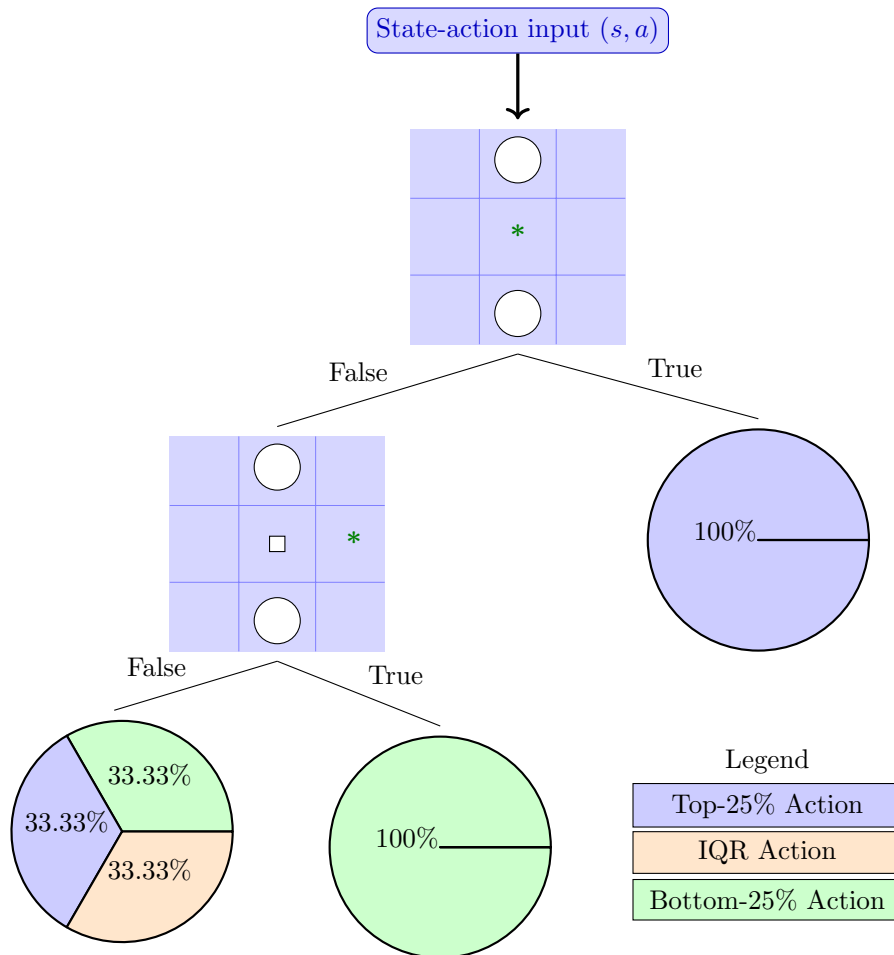


Figure 7: Example multiclass classification tree for a handcrafted policy for Tic-Tac-Toe, modelled as a decision tree that takes state-action pairs (s, a) as inputs and produces a single classification for such a pair as output. This tree can recognise some winning actions (assuming this is a policy for the white player), classifying those actions as being likely top-25% actions. It can also recognise some cases where winning actions are possible but not picked (actions placing next to an empty cell in between two white pieces), and classifies those as being likely bottom-25% actions. Any other cases are classified as being equally likely to belong to any of the three possible classes.

This is not strictly wrong: it is very well possible that certain patterns correlate strongly with both weak and strong actions, whilst not correlating strongly with “average” actions. However, it may be considered undesirable for the purpose of generating explanations of tactics, since such an output distribution does not give any clear, actionable recommendations. There are techniques to account for ordinal classes, but they work by transforming a single k -class classification problem into $k - 1$ separate binary classification problems [63]. This would result in a collection of multiple different decision trees, which would also hamper interpretability.

To avoid the potential for confusion discussed above, we propose to further simplify the output space by training a classifier that outputs a single probability estimate for any given (s, a) input pair. This can be viewed as a binary classification problem, with “positive” and “negative” classes. The first two types of target labels that were considered, but found to be problematic, are:

1. Best-action indicator, i.e. a target class of 1 if and only if $\pi(s, a) = \max_{a'} \pi(s, a')$, and 0 otherwise. The core issue with this is that the trained policies π are expected to be imperfect, and these target labels punish actions a' with probabilities $\pi(s, a')$ close to (but not equal to) the maximum too harshly: they are treated as being equal to the worst actions.
2. Probability of playing, i.e. a “soft” target class simply equal to $\pi(s, a)$. The core issue with this target label is that it is highly sensitive to the number of legal actions in a state s : the best action in a state with many legal actions may have a lower value $\pi(s, a)$ than a weak action in a different state with few legal actions.

Finally, as a target label that does not suffer from either of the issues described above, we propose to use $\frac{\pi(s, a)}{\max_{a'} \pi(s, a')}$ as the target label for an input pair (s, a) . The outputs of such a model may intuitively be interpreted as estimators of the (unnormalised, since they do not add up to 1) probabilities of actions to be the best action in their state. This is somewhat similar to the logit regression tree outputs, but the main difference is that these outputs are on a linear scale,

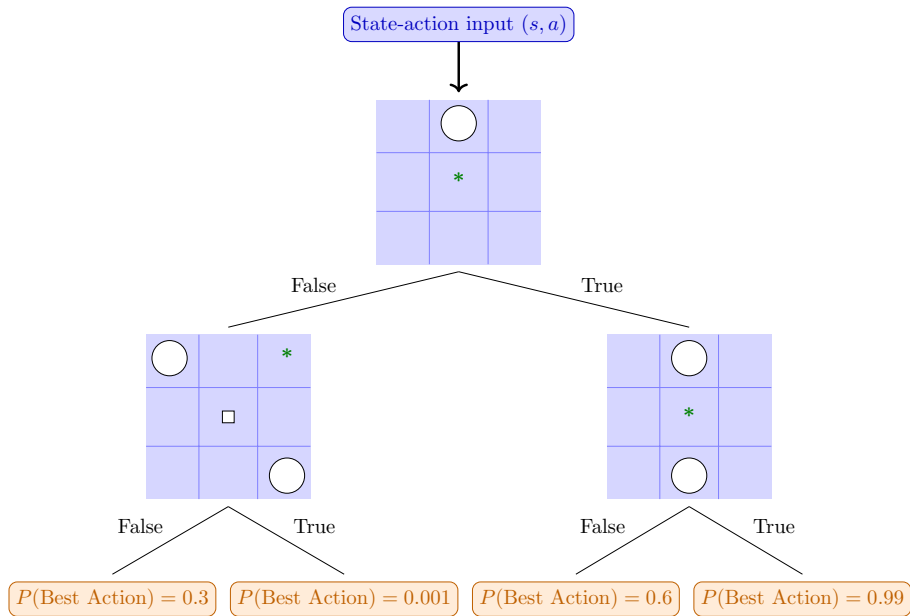


Figure 8: Example best-action classification tree for a handcrafted Tic-Tac-Toe policy, modeled as a binary classification tree that takes state-action pairs (s, a) as inputs, and produces probability estimates of a being a “best action” in s as outputs.

rather than the exponential scale on which logits should be interpreted. Figure 8 depicts a handcrafted example of this type of tree.

390 4.1.4. Imbalanced Best-Action Classification Trees

As a final type of decision tree, we consider one that uses the same target labels as described in 4.1.3, but where every branch for cases where a feature evaluates to true is forced to immediately lead to a leaf node. Only branches followed when tested features evaluate to false can lead to new decision nodes.

395 This special structure means that the decision tree may be read as a chain of *if-then-else-if* rules. These are arguably even easier to interpret than more balanced decision trees, since a human can forget about previous features when navigating down the tree (or list of rules) when “simulating” the decision tree’s process; as soon as one feature evaluates to true, it is guaranteed to immediately

400 produce an output for that input. The example decision tree depicted in Figure 8

would have qualified as this type of tree if the “True” branch from the root node directly led to a single leaf.

4.2. Training Classification and Regression Trees

Classification and regression trees are trained using the customary top-down
405 tree induction strategy that, at each branching point, greedily selects whichever
feature maximises some notion of information gain when used to split on [64].
The self-play training process used to train our initial policies [20] collects game
states s encountered during self-play between MCTS-based players in an expe-
rience buffer. These game states, extracted from the experience buffer at the
410 end of the training process, and the fully trained (linear) policy π_{θ} , are used to
construct the training data set for the decision trees.

Let \mathcal{D} denote a dataset of all state-action pairs (s, a) that can reach a node
in a decision tree for the next feature to split on when building a tree is to
be determined. For example, in the case of a root node, this would simply
415 be the set of all possible (s, a) pairs such that $a \in \mathcal{A}(s)$ is a legal action in
 s , and s is one of the game states extracted from the experience buffer. In
the case of a node deeper than the root node, this set would be reduced to
only contain those (s, a) pairs that would lead to the node under consideration,
based on the feature vector $\phi(s, a)$ and the tests performed in earlier nodes of
420 the decision tree. Let ϕ_i be a candidate feature under consideration to be split
on. Let $\mathcal{D}_{\phi_i}^T \subseteq \mathcal{D}$ denote the subset of data that would follow the branch for
 $\phi_i(s, a) = 1$, and $\mathcal{D}_{\phi_i}^F \subseteq \mathcal{D}$ the remaining subset for the case where $\phi_i(s, a) = 0$.
The following subsections describe the splitting criteria used for the various
types of decision trees proposed in this paper. For all types of decision trees,
425 splits that result in either one of the branches representing $k \leq 5$ state-action
pairs are prohibited, and splits that do not provide any improvement whatsoever
with respect to the splitting criterion in comparison to the current (unsplit) node
are also prohibited.

4.2.1. Training Logit Regression Trees

The splitting criterion used for logit regression trees is to select features that lead to a minimal sum of squared errors between the target logits $z(s, a) = \boldsymbol{\theta}^\top \boldsymbol{\phi}(s, a)$ and the logits predicted by the regression tree. It is assumed that a leaf node of a logit regression tree simply predicts the mean of the target logits $z(s, a)$ for all (s, a) pairs in the dataset that lead to that leaf. With some abuse of notation, let $\bar{z}(\mathcal{D})$ denote the mean of all the $z(s, a)$ values for all (s, a) pairs in a dataset \mathcal{D} . The sum of squared errors resulting from a split on a candidate feature ϕ_i is given by Equation 3:

$$SSE(\mathcal{D}, \phi_i) = \sum_{(s,a) \in \mathcal{D}_{\phi_i}^T} (z(s, a) - \bar{z}(\mathcal{D}_{\phi_i}^T))^2 + \sum_{(s,a) \in \mathcal{D}_{\phi_i}^F} (z(s, a) - \bar{z}(\mathcal{D}_{\phi_i}^F))^2 \quad (3)$$

430 4.2.2. Training Multiclass State-Action Classification Trees

As described in Subsubsection 4.1.2, every state-action pair (s, a) is assigned one class $c(s, a) \in \mathcal{C}$ as target label, where in this paper a set of three possible classes $\mathcal{C} = \{\text{Bottom25\%}, \text{IQR}, \text{Top25\%}\}$ is used. Let $0 \leq P(c' | \mathcal{D}) \leq 1$ denote the proportion of state-action pairs (s, a) in a dataset \mathcal{D} such that $c(s, a) = c'$. Let $|\mathcal{D}|$ denote the cardinality of a dataset \mathcal{D} , i.e. the number of state-action pairs it contains. Let $H(\mathcal{D}) = -\sum_{c' \in \mathcal{C}} P(c', \mathcal{D}) \log_2(P(c', \mathcal{D}))$ denote the entropy of a dataset \mathcal{D} . The feature ϕ_i used for splitting is the one that maximises information gain, which is computed as in Equation 4:

$$IG(\mathcal{D}, \phi_i) = H(\mathcal{D}) - \frac{|\mathcal{D}_{\phi_i}^T|}{|\mathcal{D}|} H(\mathcal{D}_{\phi_i}^T) - \frac{|\mathcal{D}_{\phi_i}^F|}{|\mathcal{D}|} H(\mathcal{D}_{\phi_i}^F) \quad (4)$$

4.2.3. Training Best-Action Classification Trees

As described in Subsubsection 4.1.3, smooth target labels $\frac{\pi(s, a)}{\max_{a'} \pi(s, a')}$ are used, rather than discrete (binary) class labels for the best-action classification trees. This means that, even though we may intuitively think of them as classifiers—due to the outputs being interpretable as estimates of the probability of belonging to a *best-action* class—they technically function more like regression trees. Therefore, a splitting criterion similar to Equation 3 is used, with $\frac{\pi(s, a)}{\max_{a'} \pi(s, a')}$ rather than logits as target labels. The key distinction with

logit regression trees is that these outputs are on a linear scale, and bounded in
440 the $[0, 1]$ range, whereas the logit outputs are on an exponential scale, with an
unbounded range.

4.2.4. Training Imbalanced Best-Action Classification Trees

The primary distinction between the imbalanced and the regular best-action
classification trees is that, in the imbalanced version, further splits are never
445 created after at least one feature has evaluated to `true`. Two variants of this
idea are considered.

The first variant, referred to as the asymmetric variant, only takes into
consideration the sum of squared errors over the subset of data $\mathcal{D}_{\phi_i}^T$ in the “true”
branch of a potential split on a feature ϕ_i . The rationale behind this is that,
450 if we read such an imbalanced tree as a chain of *if-then-else-if* rules, purity in
the subset of data that a rule applies to may be valued more highly than purity
in the other subset of data that a rule does *not* apply to. If a rule applies,
the model gives a direct recommendation, which we can be more confident in
if the dataset it applies to is pure. In contrast, if a rule does not apply, we
455 simply drop down to subsequent rules (if any exist), rather than giving a direct
recommendation.

The second variant, referred to as the symmetric variant, sums up the sums
of squared errors for both subsets of data resulting from a split, as per the
usual splitting criterion. Note that “asymmetric” and “symmetric” refer to
460 (a)symmetry in which subsets of data play a role in splitting criteria, whereas
“imbalanced” is used to describe the shape of the tree.

4.3. Policy Training Objective

All of the splitting criteria discussed previously for the various types of de-
cision trees depend on the parameters θ of a fully trained policy π_θ —either
465 through the logits $z(s, a)$ it computes for state-action pairs (s, a) , or the action
probabilities $\pi_\theta(s, a)$ computed by such a policy. A common training objective
for training such a policy from self-play, following Expert Iteration [65] and Al-

phaGo Zero [2], is to minimise the cross-entropy (CE) between the policy π and an expert policy π^M , which is typically derived from the distribution of visit
470 counts of a tree search process by MCTS.

Soemers et al. [66] remarked that MCTS (by design) allocates a part of its search budget on exploration, and that this means that a policy π trained to mimic the behaviour of MCTS through such a CE-based objective is also explicitly trained to have some degree of exploratory behaviour. While this is
475 desirable when such a policy is subsequently used to guide future tree searches (which should again have some degree of exploration), it may be less desirable for extracting explainable tactics or a small set of key features. In comparison to CE, an alternative training objective referred to as Tree-Search Policy Gradients (TSPG) [66] was shown to (i) produce policies that are stronger in terms of
480 standalone playing strength (without tree search), (ii) have a more precise focus with larger weights distributed over a smaller set of features, and (iii) have less entropy in the resulting probability distributions over actions. Due to these aspects, the TSPG objective was hypothesised to be more suitable than CE for goals such as the ones considered in this paper. To further evaluate this,
485 decision trees trained on policies optimised for TSPG as well as the standard CE objective are included in the following experiments.

5. Evaluation

For a quantitative empirical evaluation, we focus on comparing the playing strength of the various types of decision trees proposed in this paper to that
490 of the full policies (using all discovered features). This is comparable to the experiments used in other work on explainable RL based on various types of decision trees and rules [40, 37, 38, 39], and can also give an indication of whether or not the trees successfully select and focus on the most important features. The following types of trained agents are considered:

- 495 • **Logit (Obj; d)**: logit regression tree (see 4.1.1) with a maximum depth of d , trained to mimic the full policy with objective **Obj** (either CE or

TSPG).

- **Multiclass (Obj; d):** multiclass state-action classification tree (see 4.1.2) with a maximum depth of d , trained to predict between three classes (bottom 25%, IQR, top 25%), based on the full policy with objective **Obj** (either CE or TSPG).
500
- **Best-Action (Obj; d):** best-action classification tree (see 4.1.3) with a maximum depth of d , trained for binary classification (output probability of being best action), based on the full policy with objective **Obj** (either CE or TSPG).
505
- **Asymm. Imb. Best-Action (Obj; d):** imbalanced best-action classification tree (see 4.1.4) with a maximum depth of d , with imbalanced tree structure and asymmetric splitting criterion (see 4.2.4), based on the full policy with objective **Obj** (either CE or TSPG).
- **Symm. Imb. Best-Action (Obj; d):** imbalanced best-action classification tree (see 4.1.4) with a maximum depth of d , with imbalanced tree structure and symmetric splitting criterion (see 4.2.4), based on the full policy with objective **Obj** (either CE or TSPG).
510
- **Full Policy (CE):** the full (linear) policy trained for the standard Cross-Entropy (CE) objective, using all discovered features.
515
- **Full Policy (TSPG):** the full (linear) policy trained for the Tree-Search Policy Gradients (TSPG) objective [66], using all discovered features.

Unless specified otherwise, these agents select actions as follows. The agents based on logit regression trees sample actions according to a softmax over the output logits from their trees. The agents based on multiclass classification trees sample actions proportionally to $P(\text{Top 25\%}) \times (1 - P(\text{Bottom 25\%}))$. The agents based on any of the best-action classification trees sample actions proportionally to their outputs. The full (linear) policies sample actions accord-
520

ing to a softmax over the logits predicted by their dot products. Additionally,
525 two types of agents that do not involve any training are included:

- **Random:** an agent that selects actions uniformly at random.
- **UCT:** a standard UCT agent [23, 25], using 1 second of thinking time per move (note that all other agents play significantly faster than this, since they do not run any tree search).

530 All agents, training code, games, and experiments are implemented in the Ludii general game system [52, 53].²

5.1. Results in Small Games

First, results are presented from experiments in a set of thirteen “small games.” These are sequential, deterministic, 2-player games played on relatively
535 small boards—each having at most eleven playable sites. In all of these, basic tree search algorithms such as UCT, and potentially even trained policies based on simple features, may be expected to be capable of strong or even optimal play. The games included in this set are *Akidada*, *Alquerque de Tres*, *Haretavl*, *Hat Diviyay Keliya*, *Ho-Bag Gonu*, *Jeu Militaire*, *Kaooa*, *Madelinette*,
540 *Mu Torere* (with the *Complete (Observed)* ruleset), *Mu Torere* (with the *Simple (Suggested)* ruleset), *Pong Hau K’i*, *Three Men’s Morris*, and *Tic-Tac-Toe*.

For each of these games, every type of decision tree is trained with maximum depths of $d \in \{1, 2, 3, 4, 5, 10\}$. This means that we ultimately end up with
545 $(5 \times 2 \times 6) + 4 = 64$ distinct agents: 5 types of decision trees, each trained for 2 objectives (CE and TSPG), each with 6 different depth limits, plus the 2 full policies (CE and TSPG), the random agent, and the UCT agent. Each of these agents is evaluated in every game by playing 50 matches (25 as first and 25 as second player) against each of the 63 other agents, for a total of $50 \times 63 = 3,150$ matches per game, per agent. Win percentages averaged over
550 all possible opponents in a game are used as the primary measure of playing

²Source code is available at <https://github.com/Ludeme/Ludii/>.

strength. Draws are counted as half wins for each player. If a match did not end after 250 moves, it is declared a draw.

Figure 9 depicts the average win percentages of all of these agents, over all thirteen small games, on the y -axis. The maximum depth d for agents based on decisions trees is varied along the x -axis. The four agents that are not based
555 on decision trees are simply plotted as horizontal lines. Policies optimised for CE (as well as Random) are drawn as dotted lines, and policies optimised for TSPG are (as well as UCT) are drawn as solid lines.

To summarise the results across all thirteen small games in a single plot,
560 performance profiles [67] are provided for all types of agents (only displaying decision tree agents with $d = 5$) in Figure 10. The x -axis shows average UCT-normalised scores, which are scores (win percentages) that have been linearly rescaled on a per-game, per-opponent basis, such that 1.0 corresponds to the performance of UCT in that game against that opponent. The y -axis shows the
565 fraction of runs for which an agent obtained a score greater than any given UCT-normalised score τ . Shaded areas indicate 95% bootstrap confidence intervals based on 10,000 bootstrap replicates, sampling from the runs against different opponents. Note that this means that the intervals indicate uncertainty due to variability in performance with respect to different opponents, rather than
570 variability due to randomness in any training or evaluation processes.

5.2. Results in Other Games

Where the results described above were for a set of thirteen small games, in this section we look towards a different set of thirty other games: *Alquerque*,
Amazons, *Ard Ri*, *Arimaa*, *Ataxx*, *Bao Ki Arabu (Zanzibar 1)*, *Bizingo*, *Break-*
575 *through*, *Chess*, *English Draughts*, *Fanorona*, *Fox and Geese*, *Go*, *Gomoku*,
Gonnect, *Havannah*, *Hex*, *Knightthrough*, *Konane*, *Lines of Action*, *Omega*,
Pentalath, *Pretwa*, *Reversi*, *Royal Game of Ur*, *Shobu*, *Surakarta*, *Tablut*, *XII Scripta*, and *Yavalath*. All of these are sequential, 2-player games, with most of them being deterministic, but some stochastic. In contrast to the small games,
580 plain UCT agents or trained policies using only simple patterns cannot be ex-

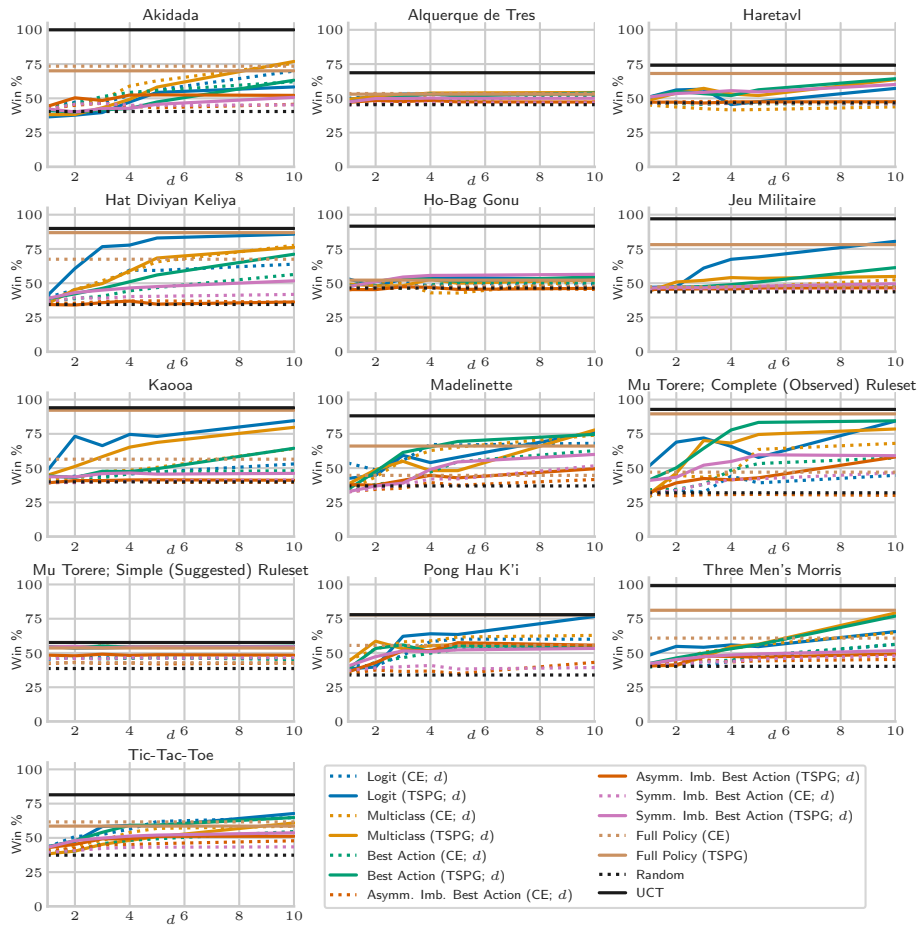


Figure 9: Win percentages, averaged over all other agents as opponents, for various types of agents in thirteen small games. Data plotted for maximum decision tree depths $d \in \{1, 2, 3, 4, 5, 10\}$ along x -axis.

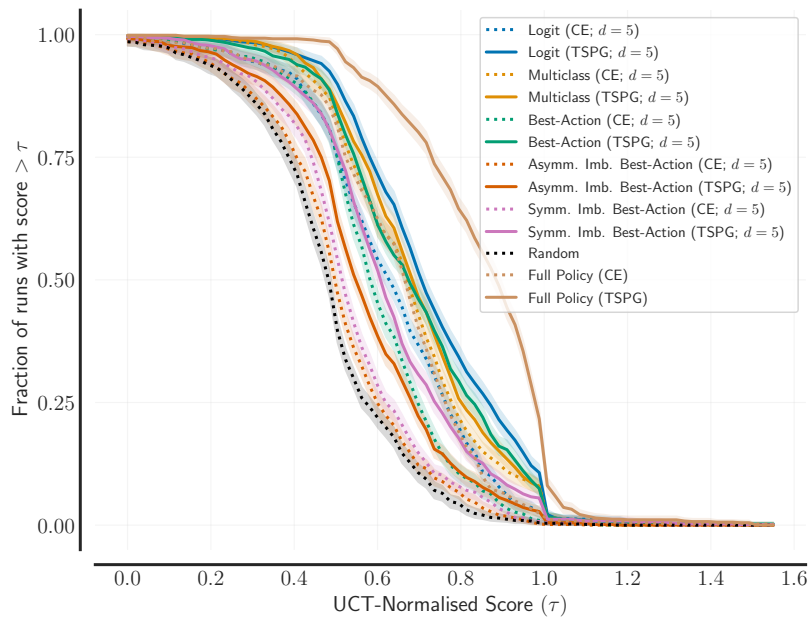


Figure 10: Performance profiles [67] for several agents, summarising performance across all thirteen small games. Performance is measured by UCT-normalised scores τ , which are win percentages that have been linearly rescaled on a per-game, per-opponent basis, with the performance of UCT always being equal to 1.0.

pected to play (close to) optimally in these games, but meaningful (better than random) play may still be expected.

The overall setup of experiments in these games is similar to the setup described above for small games, with two primary differences. Firstly, UCT is
585 no longer included in the evaluations in order to avoid the large amount of computation time required by this agent for evaluations against a large number of possible opponents (63 other agents) in such a large set of games, including many relatively large and complex games. Secondly, matches are allowed to continue for up to 1,000 moves rather than 250 before declaring them a draw,
590 since the limit of 250, which is appropriate for the small games, may be too low for many of these larger games.

Figure 11 summarises the results for all 63 agents by reporting the median, interquartile mean, and mean win rate for every agent against all other agents, across all thirty games. The 95% bootstrap confidence intervals represent vari-
595 ability in performance across different games and different opponents, rather than variability due to stochasticity in training or evaluation processes. Figure 12 depicts performance profiles for the same experiment, restricted to only depth limits of $d = 5$ for agents based on decision trees for visual clarity.

5.3. *Biasing MCTS with Trained Features*

600 While previous work has already demonstrated that full sets of features (without subsequent selection of a smaller subset) can improve the playing strength of MCTS by biasing it in many games, it was also found that they can reduce the playing strength in some games [21, 22]. Such reductions in playing strength are likely due to the computational overhead incurred by us-
605 ing features, which may be mitigated by using only smaller subsets of features. Therefore, the performance of biased versions of MCTS—biased by full policies as well as decision trees—is evaluated against the standard UCT baseline, on the complete set of thirty games also used in the previous subsection. Every agent uses one second of search time per move. For every game and every
610 matchup, 150 evaluation matches were run (with every agent playing each side



Figure 11: Aggregate metrics for the performance levels of all 63 agents in the set of thirty games. Median, interquartile mean (IQM), and mean of the win rates are computed over runs in all games against all possible opponents. Coloured bars represent 95% bootstrap confidence intervals, estimated from 10,000 bootstrap replicates using the `rliable` library [67].

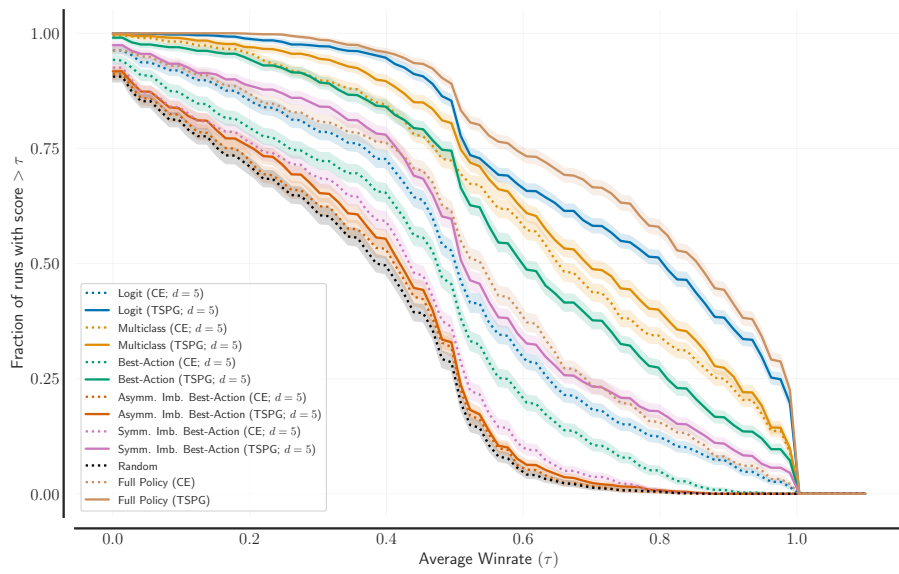


Figure 12: Performance profiles summarising the performance of several agents (only displaying results for decision trees with depth limits of $d = 5$) across thirty different games. Performance is measured by the win rate averaged over all possible opponents (all other agents, including trees with depth limits $d \neq 5$).

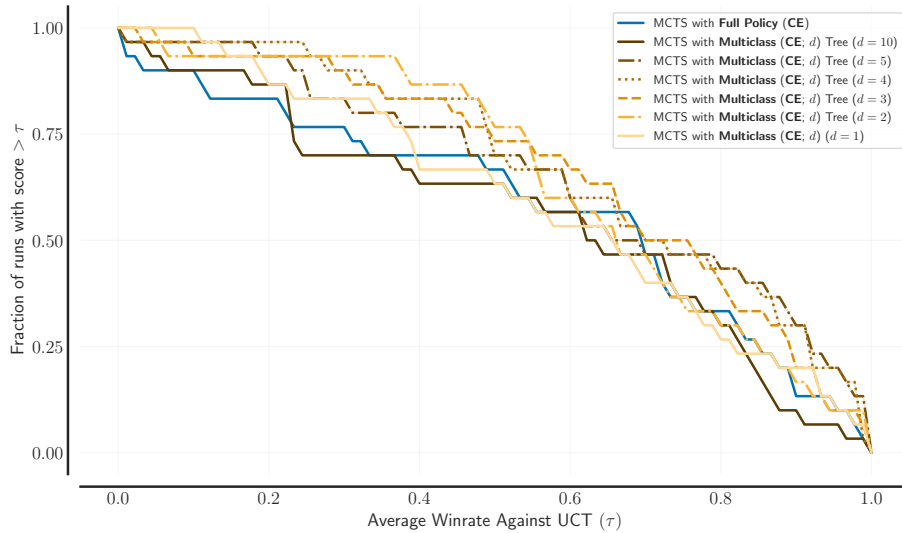


Figure 13: Performance profiles summarising the performance of MCTS agents biased by several different policies against UCT, across thirty different games.

of the matchup 75 times).

Because policies trained for the TSPG objective were previously found not to provide additional value to MCTS [66], we focus only on policies trained for the CE objective. This experiment is repeated with the **Logit** trees (because their output representation is the same as that of the full policies), and **Multiclass** trees (because **Multiclass (CE; 5)** appears to outperform **Logit (CE; 5)** in Figure 12).

Table 2 lists the median, interquartile mean (IQM), and mean win rates of MCTS agents biased by various different policies against UCT, aggregated over the thirty games. Figure 13 additionally depicts performance profiles for MCTS agents biased by **Multiclass** trees (as well as the full policy) against UCT. Performance profiles for agents biased by **Logit** trees are omitted to save space (these results were similar to those for the **Multiclass** trees).

Agent	Win rate against UCT		
	Median	IQM	Mean
MCTS with Logit (CE; d) trees			
$d = 1$	0.61	0.66	0.64
$d = 2$	0.72	0.70	0.68
$d = 3$	0.69	0.71	0.70
$d = 4$	0.70	0.70	0.69
$d = 5$	0.63	0.66	0.66
$d = 10$	0.67	0.62	0.59
MCTS with Multiclass (CE; d) trees			
$d = 1$	0.66	0.62	0.60
$d = 2$	0.66	0.67	0.66
$d = 3$	0.73	0.72	0.67
$d = 4$	0.72	0.72	0.68
$d = 5$	0.67	0.71	0.66
$d = 10$	0.63	0.60	0.56
MCTS with Full Policy (CE)			
	0.69	0.64	0.58

Table 2: Median, interquartile mean (IQM), and mean win rates (across thirty games) of MCTS agents biased by various trained policies against UCT.

5.4. Discussion

625 The plots in Figure 9 show that, generally, the playing strength tends to increase (or eventually stabilise) as the depth limits of decision trees increase in the small games. As expected, deeper decision trees can express more powerful policies. The imbalanced trees tend to perform worse than the (larger) balanced trees, and trees (as well as non-tree policies) trained for the TSPG objective tend
630 to outperform policies trained for the CE objective. These plots do not provide a clear ranking among the other types of decision trees (Logit, Multiclass, and Best-Action), since this differs from game to game.

Three of the thirteen small games appear to be exceptions, in that most types of policies have similar levels of playing strength, and these remain constant
635 regardless of depth limits. This may be an indication that these games have a low strategic (or tactical) depth [68], but it may also simply indicate that the training algorithms fail to learn relevant tactics. For the first of these three games, *Alquerque de Tres*, an $\alpha\beta$ -search [69] of less than a second easily finds that optimal play leads to a draw after six moves, which indeed points to a
640 game with relatively little strategic depth. The second game, *Ho-Bag Gonu*, keeps going on infinitely under perfect play, and requires a chain of multiple unforced errors before an optimal player can capitalise and obtain a victory. The third game, *Mu Torere; Simple (Suggested Ruleset)* uses a flawed (as a result of a mistranslation) ruleset [70, 15] in which the first player can win in
645 a single move. This is in contrast to *Mu Torere; Complete (Observed Ruleset)*, which uses the ruleset based on the correct translation, for which a greater variety in performance levels between the policies is observed. For all three of these games, we find that sufficiently deep trees can learn to play optimally against UCT, but they fail to learn how to exploit mistakes by suboptimal
650 players. In the case of *Mu Torere*, no features or trees are learnt for the second player at all, because all the experience collected from self-play by MCTS-based agents consists of the first player winning in a single move. These issues could potentially be improved by introducing additional exploration in the self-play process.

655 The performance profiles depicted in Figure 10 suggest that, on average,
Logit outperforms Multiclass (for the TSPG objective), and Multiclass outper-
forms Best-Action, followed by the Symmetric and Asymmetric variants of the
Imbalanced Best-Action classification trees. This ordering corresponds to the
number of simplifications and approximations made for the sake of arguable
660 improvements with respect to interpretability. The output type of Logit trees is
equal to the output type of the original policies, but difficult to interpret. The
Multiclass classification tree simplifies the output representation to a three-
class problem, and the Best-Action classification tree further simplifies this to a
binary classification problem. The Symmetric Imbalanced Best-Action tree im-
665 poses additional constraints on the shape of the tree (also causing it to use fewer
features), and the Asymmetric variant furthermore adjusts the splitting crite-
rion. However, these differences in performance tend to be relatively small, in
particular among the top three decision tree types. One exception is that, when
training for the cross-entropy objective, Multiclass trees appear to outperform
670 Logit trees.

For the set of larger games, Figures 11 and 12 paint a similar overall picture
in terms of ranking decision tree types by playing strength, albeit with more
pronounced differences between the types. Policies trained for the TSPG objec-
tive also have a clearer advantage over policies trained for CE, with Figure 11
675 even showing that some TSPG trees limited to a depth of $d = 1$ perform at a
similar level to CE trees limited to a depth of $d = 10$. Again, Multiclass trees
trained for the CE objective outperform Logit trees for this objective. We hy-
pothesise that using the less fine-grained output representation of the Multiclass
trees may implement a helpful form of regularisation.

680 Table 3 provides upper bounds on the numbers of distinct features that
various types of policies may use. The full policy always has up to 400 new
features generated from self-play (but possibly fewer if the training process
takes too long). The number of (atomic) features that are used to initiate
a training process can vary greatly depending on the game. We focused on
685 displaying and discussing results for the trees limited to $d = 5$, since these are

Policy	Number of Distinct Features		
	Initial Feature Set	From Training	Total
Balanced Decision Trees			
$d = 1$			1
$d = 2$			≤ 3
$d = 3$			≤ 7
$d = 4$			≤ 15
$d = 5$			≤ 31
$d = 10$			≤ 1023
Imbalanced Decision Trees			
$d = 1$			1
$d = 2$			≤ 2
$d = 3$			≤ 3
$d = 4$			≤ 4
$d = 5$			≤ 5
$d = 10$			≤ 10
Full Policy	72–452	≤ 400	≤ 472 –852

Table 3: Upper bounds on the numbers of distinct features that various policies may use.

guaranteed to use substantially fewer features than the full policy. The larger trees limited to $d = 10$ might use the full feature set, which makes evaluating them less interesting with respect to the potential use of decision trees for feature selection.

690 The results in Table 2 show that, according to all of the three different aggregate statistics (median, IQM, and mean), MCTS agents biased by any of the trained policies tend to outperform UCT over the set of thirty games. Note that there may of course be individual games where this is not the case. In general, it appears that the best results tend to be obtained by using trees
695 limited to depths of 3 or 4. The observation that such policies tend to outperform the larger trees and the full policies suggests that feature selection can indeed improve playing strength, likely due to a reduction in computational overhead.

The performance profiles in Figure 13 tend to intersect many times. This suggests that there is no clear single restriction on the size of trained policies that
700 consistently works best for biasing MCTS in all games. Only the full policies and the largest trees, with a depth limit of $d = 10$, somewhat stand out as likely being the worst performers. In the top-left section of the plot, the smallest tree restricted to $d = 1$ has the best performance level; this policy (with the lowest computational overhead) has the lowest number of cases with an extremely
705 poor performance level. In the bottom-right section of the plot, the trees with depth limits of 4 and 5 have the best performance levels; these policies have the greatest likelihood of delivering extremely strong levels of performance (win rates exceeding 0.8).

5.5. Case Studies

710 In addition to the quantitative results focused on playing strength, we manually inspect several decision trees and the features they use for two different games. This gives an impression of the ways in which we can learn about the games' tactics as well as the AI training process.

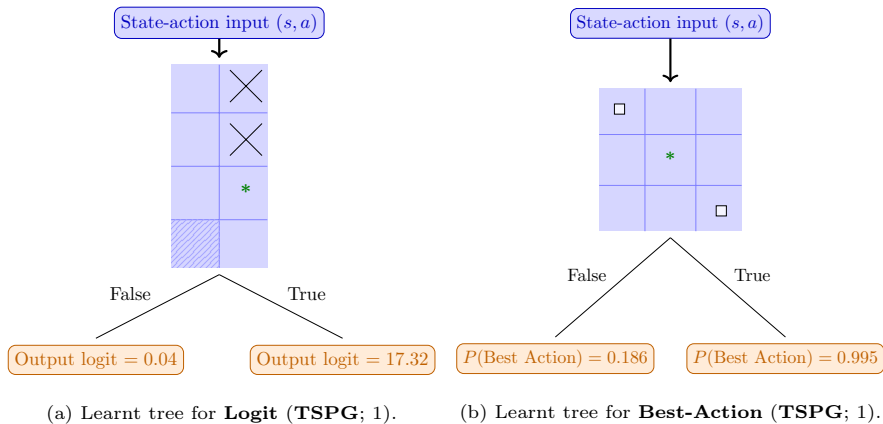


Figure 14: Two trees with depth limits of $d = 1$ that were built for the game of Tic-Tac-Toe. A shaded square indicates that that position must be an “off-board” position (i.e., not exist as a part of the board). A small white square indicates a position that must be empty.

5.5.1. Tic-Tac-Toe

715 Figure 14 depicts two trees, each restricted to a maximum depth of $d = 1$, that were learnt for Player 1 (the white player) for the game of *Tic-Tac-Toe*. The tree in Figure 14(a) is a Logit regression tree, and the one in Figure 14(b) is a Best-Action classification tree. Due to the depth limit of $d = 1$, each tree is limited to only a single feature.

720 Both trees have selected sensible features that are clearly relevant to the game, but different ones. The Logit regression tree has selected a feature that strongly recommends playing below a consecutive line of two crosses, which prevents the opponent from making a winning move in the next turn. Note that, due to rotations and reflections, this feature can also apply to moves that
725 block orthogonal lines by placing a circle to their left, right, or above them. However, blocking lines by placing a stone in between two opposing pieces, or blocking diagonal lines, would require additional features. It should be remarked that this feature also has a redundant constraint: it requires a hypothetical site diagonally below the recommended action to be off the board. Because the
730 game is played on a 3×3 grid of square cells, the constraint is already implied by the rest of the feature. Hence, an equivalent feature (which would likely

be preferable from the point of view of interpretability) could simply omit this constraint. This equivalence is emergent from the game’s rules, as discussed in 3.2.

735 The feature used by the Best-Action classification tree is one that recommends playing in a position that has two opposite diagonal connections to empty cells. On the 3×3 grid of Tic-Tac-Toe, the only cell that can ever satisfy this condition is the centre cell. This feature no longer recommends playing in the centre if at least two corners on the same side of the board are already occupied,
740 so in theory it is different from a pure centre-detector. However, in practice, it is almost equivalent due to the strong preference of many agents—such as MCTS agents, but also the policy of this decision tree itself—to immediately play in the centre in the very first move of the game. This is an example of a dependence between features that emerges from the policies used to play, as
745 discussed in 3.3.

5.5.2. *Hex*

Hex is a 2-player connection game, played on a (by default 11×11) rhombus of hexagonal cells, where each player has the objective of creating a connection between two opposite sides of the board with pieces of their colour. For this
750 game, there is extensive documentation available of patterns, tactics, and strategies that work well for humans [71, 72] as well as AI players [49, 50]. Figure 15 depicts several features for this game.

The feature in Figure 15(a) is selected as the first feature by **Logit (TSPG; d)** trees. In the case of such a tree restricted to $d = 1$, it predicts a logit of 4.16
755 for actions that match the feature, and 25.86 otherwise, which means that it strongly discourages such moves. We are not aware of this having any particular strategic or tactical relevance, and suspect it is simply an artifact resulting from the limited playing strength of the MCTS agents used in self-play training.

The feature in Figure 15(b) is selected as the first feature by **Multiclass (TSPG; d)** trees. The tree restricted to $d = 1$ predicts $\{P(\text{Bottom } 25\%) = 3.7\times 10^{-4}, P(\text{IQR}) = 0.98, P(\text{Top } 25\%) = 0.02\}$ for actions that match the feature,

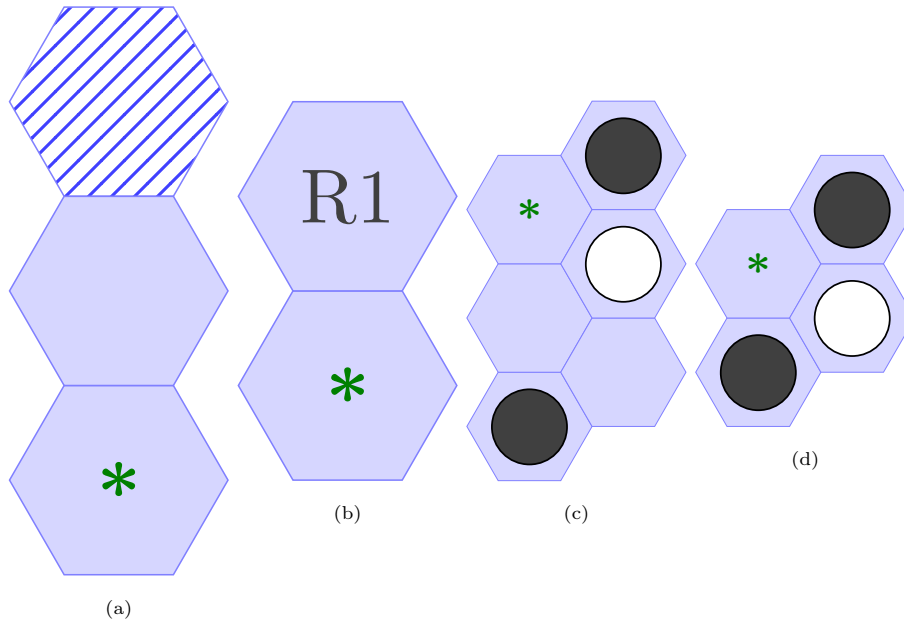


Figure 15: Four features for the first player (white) in the game of *Hex*. (a) The first feature selected by **Logit (TSPG; d)** trees. It matches any move in the second ring of cells from the edge of the board. (b) The first feature selected by **Multiclass (TSPG; d)** trees. It matches any move that has an adjacent position which is closer to the region with index 1 (which, in Ludii, is the set of all cells along the black edges, which the black player aims to connect) than the position of the move. In practice, this feature applies to any move except for moves along the black board edges. (c) The first feature selected by **Best-Action (TSPG; d)** trees, including imbalanced variants. Note that the undecorated cells are unrestricted; they need not be empty. (d) A bridge intrusion feature, which is a well-known tactic for *Hex* to stop the two black stones from connecting.

and $\{P(\text{Bottom } 25\%) = 0.07, P(\text{IQR}) = 0.93, P(\text{Top } 25\%) = 4.1 \times 10^{-4}\}$ for actions that do not match the feature. This means that the feature is used to slightly discourage playing in either of the opponent’s goal regions, which
765 may indeed be a useful basic strategy for AI. However, the fact that actions are classified as being overwhelmingly likely to belong to the IQR class regardless of whether or not the feature matches, with probabilities of 0.98 and 0.93, respectively, arguably makes for questionable advice to humans.

The feature in Figure 15(c) is selected as the first feature by **Best-Action**
770 (**TSPG**; d) trees. The tree restricted to $d = 1$ predicts a probability of 0.22 of being the best action for actions that match the feature, versus a probability of 0.01 for actions that do not match the feature. The action encouraged by this feature prevents what could otherwise become a future connection between the two opposing (black) pieces. It could be viewed as a longer-range, less
775 “urgent” variant of the feature depicted by Figure 15(d), which in turn closely relates to the well-known concepts of virtual connections and bridges in Hex and many other connection games [71, 72]. Patterns related to this strategy have also previously been used to improve the playing strength of UCT in several connection games, including Hex [49]. The feature depicted in Figure 15(c) is
780 also used by the other types of decision trees at deeper levels; at depth $d = 3$ for the Logit regression tree, and depth $d = 2$ for the Multiclass classification tree. Note that the formation of the two black pieces in Figure 15(c) would, in the absence of other pieces, have formed a *loose connection*, which Browne [71] describes as an intermediate Hex strategy. Hence, this feature could also
785 be viewed as the final step of an attack against such a loose connection by the white player.

6. Conclusion

We have explored the problem of distilling linear policies for game-playing—
in this case based on local patterns that are matched to the game state in the
790 spatial area around actions—into various forms of decision trees, for a wide vari-

ety of board games. Decision trees inherently perform feature selection, meaning that they can be used to identify key features from a substantially larger set of features used in the original policies. Furthermore, decision trees are generally considered to be highly interpretable models, making them a popular choice
795 for explaining policies in RL. Many of the games considered in this paper have substantially larger action spaces than environments used in previous work on explaining policies. Therefore, we proposed and evaluated a variety of different output representations for decision trees that take state-action pairs, rather than just states, as inputs. These provide per-action advice in a variety of forms,
800 with some hypothesised to be more readily interpretable than others.

Empirical evaluations in a set of thirteen small games, and an additional set of thirty larger games, show that the performance (in terms of playing strength) of different types of decision trees tends to decrease as the output representations are simplified for the sake of (hypothesised) improvements in interpretability,
805 but this differs from game to game. The Logit regression trees in particular tend to perform close to the level of the full policies, with significantly lower total feature counts (for depth limits up to and including $d = 5$) and lower feature counts along any single path from root to leaf. Decision trees trained for the Tree-Search Policy Gradients objective [66] convincingly outperform those
810 with equal feature counts and depth limits trained for the standard cross-entropy objective. Full policies as well as all sizes of decision trees are found to be capable of improving the playing strength of MCTS agents in expectation over thirty different games. Mid-sized trees (for the Multiclass type and CE objective), with depth limits of about 3 or 4, appear to provide the most stable improvements.
815 This suggests that using these decision trees for feature selection can improve the playing strength of biased MCTS agents. Taking two games (Tic-Tac-Toe and Hex) as case studies in which we manually inspect the learnt trees and the features they focus on, we find primarily features that are easily recognised as relevant to the games’ strategies and tactics, with a small number of features
820 for which their importance or relevance is not immediately clear to us.

In this work, feature selection (by building decision trees) was performed

as a separate step after fully training a policy. During that original policy training process, the set of features only ever grows. A core reason for not already removing any features during that process is that “irrelevant” features
825 may still be useful for building compound features, and therefore there is a risk that removing features too early hampers the ability to construct valuable new features too much. An interesting avenue for future research would be to interleave feature removal in that process in a manner that accounts for such a risk. Given the ability to obtain small sets of key features, another interesting
830 direction for future work would be to use such small sets of features in move hash codes for various enhancements of tree search algorithms, such as FAST [73], PPAF [74], or move groups [75, 76, 77].

Acknowledgements

This research is funded by the European Research Council as part of the
835 Digital Ludeme Project (ERC Consolidator Grant #771292) led by Cameron Browne at Maastricht University’s Department of Data Science and Knowledge Engineering. This work was carried out on the Dutch national e-infrastructure with the support of SURF Cooperative. We wish to thank Cameron Browne and Mark Winands for feedback on earlier drafts of the paper. We thank the
840 anonymous reviewers for helpful feedback on the paper.

References

- [1] D. Silver, A. Huang, C. Maddison, A. Guez, L. Sifre, G. van den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, S. Dieleman, D. Grewe, J. Nham, N. Kalchbrenner, I. Sutskever, T. Lillicrap,
845 M. Leach, K. Kavukcuoglu, T. Graepel, D. Hassabis, Mastering the game of Go with deep neural networks and tree search, *Nature* 529 (2016) 484–489.
- [2] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, Y. Chen, T. Lillicrap, F. Hui,

- 850 L. Sifre, G. van den Driessche, T. Graepel, D. Hassabis, Mastering the
game of Go without human knowledge, *Nature* 550 (2017) 354–359.
- [3] M. Moravčík, M. Schmid, N. Burch, V. Lisỳ, D. Morrill, N. Bard, T. Davis,
K. Waugh, M. Johanson, M. Bowling, Deepstack: Expert-level artificial
intelligence in heads-up no-limit poker, *Science* 356 (2017) 508–513.
- [4] N. Brown, T. Sandholm, Superhuman AI for heads-up no-limit poker:
855 Libratus beats top professionals, *Science* 359 (2017) 418–424.
- [5] D. Silver, T. Hubert, J. Schrittwieser, I. Antonoglou, M. Lai, A. Guez,
M. Lanctot, L. Sifre, D. Kumaran, T. Graepel, T. Lillicrap, K. Simonyan,
D. Hassabis, A general reinforcement learning algorithm that masters chess,
shogi, and Go through self-play, *Science* 362 (2018) 1140–1144.
- 860 [6] OpenAI, C. Berner, G. Brockman, B. Chan, V. Cheung, P. Dębiak, C. Den-
nison, D. Farhi, Q. Fischer, S. Hashme, C. Hesse, R. Józefowicz, S. Gray,
C. Olsson, J. Pachocki, M. Petrov, H. Pondé de Oliveira Pinto, J. Raiman,
T. Salimans, J. Schlatter, J. Schneider, S. Sidor, I. Sutskever, J. Tang,
F. Wolski, S. Zhang, Dota 2 with large scale deep reinforcement learning,
865 <https://arxiv.org/abs/1912.06680>, 2019.
- [7] N. Brown, T. Sandholm, Superhuman AI for multiplayer poker, *Science*
365 (2019) 885–890.
- [8] O. Vinyals, I. Babuschkin, W. M. Czarnecki, M. Mathieu, A. Dudzik,
J. Chung, D. H. Choi, R. Powell, T. Ewalds, P. Georgiev, J. Oh, D. Horgan,
870 M. Kroiss, I. Danihelka, A. Huang, L. Sifre, T. Cai, J. P. Agapiou, M. Jader-
berg, A. S. Vezhnevets, R. Leblond, T. Pohlen, V. Dalibard, D. Budden,
Y. Sulsky, J. Molloy, T. L. Paine, C. Gulcehre, Z. Wang, T. Pfaff, Y. Wu,
R. Ring, D. Yogatama, D. Wünsch, K. McKinney, O. Smith, T. Schaul,
T. Lillicrap, K. Kavukcuoglu, D. Hassabis, C. Apps, D. Silver, Grandmas-
875 ter level in StarCraft II using multi-agent reinforcement learning, *Nature*
575 (2019) 350–354.

- [9] T. Cazenave, Y.-C. Chen, G. Chen, S.-Y. Chen, X.-D. Chiu, J. Dehos, M. Elsa, Q. Gong, H. Hu, V. Khalidov, C.-L. Li, H.-I. Lin, Y.-J. Lin, X. Martinet, V. Mella, J. Rapin, B. Roziere, G. Synnaeve, F. Teytaud, O. Teytaud, S.-C. Ye, Y.-J. Ye, S.-J. Yen, S. Zagoruyko, Polygames: Improved zero learning, *ICGA Journal* 42 (2020) 244–256.
- [10] A. Fickinger, H. Hu, B. Amos, S. Russell, N. Brown, Scalable online planning via reinforcement learning fine-tuning, in: *Advances in Neural Information Processing Systems* 34, 2021.
- [11] Y. LeCun, Y. Bengio, G. Hinton, Deep learning, *Nature* 521 (2015) 436–444.
- [12] R. S. Sutton, A. G. Barto, *Reinforcement Learning: An Introduction*, 2 ed., MIT Press, Cambridge, MA, 2018.
- [13] N. C. Thompson, K. Greenewald, K. Lee, G. F. Manso, The computational limits of deep learning, <https://arxiv.org/abs/2007.05558>, 2020.
- [14] J. S. Obando-Ceron, P. S. Castro, Revisiting rainbow: Promoting more insightful and inclusive deep reinforcement learning research, in: M. Meila, T. Zhang (Eds.), *Proceedings of the 38th International Conference on Machine Learning*, PMLR, 2021, pp. 1373–1383.
- [15] C. Browne, Modern techniques for ancient games, in: *IEEE Conference on Computational Intelligence and Games*, IEEE Press, Maastricht, 2018, pp. 490–497.
- [16] C. Browne, D. J. N. J. Soemers, É. Piette, M. Stephenson, M. Conrad, W. Crist, T. Depaulis, E. Duggan, F. Horn, S. Kelk, S. M. Lucas, J. P. Neto, D. Parlett, A. Saffidine, U. Schädler, J. N. Silva, A. de Voogt, M. H. M. Winands, *Foundations of Digital Archæoludology*, Technical Report, Schloss Dagstuhl Research Meeting, Germany, 2019.

- [17] Y. Zhang, P. Tiño, A. Leonardis, K. Tang, A survey on neural network interpretability, *IEEE Transactions on Emerging Topics in Computational Intelligence* 5 (2021) 726–742. 905
- [18] G. Ras, N. Xie, M. van Gerven, D. Doran, Explainable deep learning: A field guide for the uninitiated, *Journal of Artificial Intelligence Research* 73 (2022) 329–397.
- [19] C. Browne, D. J. N. J. Soemers, E. Piette, Strategic features for general games, in: *Proceedings of the 2nd Workshop on Knowledge Extraction from Games (KEG)*, 2019, pp. 70–75. 910
- [20] D. J. N. J. Soemers, É. Piette, M. Stephenson, C. Browne, Spatial state-action features for general games, <https://arxiv.org/abs/2201.06401>, 2022.
- [21] D. J. N. J. Soemers, É. Piette, C. Browne, Biasing MCTS with features for general games, in: *Proceedings of the 2019 IEEE Congress on Evolutionary Computation*, IEEE, 2019, pp. 442–449. 915
- [22] D. J. N. J. Soemers, É. Piette, M. Stephenson, C. Browne, Manipulating the distributions of experience used for self-play learning in expert iteration, in: *Proceedings of the 2020 IEEE Conference on Games*, IEEE, 2020, pp. 245–252. 920
- [23] L. Kocsis, C. Szepesvári, Bandit based Monte-Carlo planning, in: J. Fürnkranz, T. Scheffer, M. Spiliopoulou (Eds.), *Machine Learning: ECML 2006*, volume 4212 of *Lecture Notes in Computer Science (LNCS)*, Springer, Berlin, Heidelberg, 2006, pp. 282–293. 925
- [24] R. Coulom, Efficient selectivity and backup operators in Monte-Carlo tree search, in: H. J. van den Herik, P. Ciancarini, H. H. L. M. Donkers (Eds.), *Computers and Games*, volume 4630 of *LNCS*, Springer Berlin Heidelberg, 2007, pp. 72–83.

- 930 [25] C. Browne, E. Powley, D. Whitehouse, S. Lucas, P. I. Cowling, P. Rohlfshagen, S. Tavener, D. Perez, S. Samothrakis, S. Colton, A Survey of Monte Carlo Tree Search Methods, *IEEE Transactions on Computational Intelligence and AI in Games* 4 (2012) 1–49.
- [26] C. Molnar, G. Casalicchio, B. Bischl, Quantifying model complexity via
935 functional decomposition for better post-hoc interpretability, in: P. Cellier, K. Driessens (Eds.), *ECML PKDD 2019: Machine Learning and Knowledge Discovery in Databases*, volume 1167 of *Communications in Computer and Information Science*, Springer, Cham, 2020, pp. 193–204.
- [27] C. Molnar, *Interpretable Machine Learning*, Self-published, 2020.
- 940 [28] G. Hooker, L. Mentch, S. Zhou, Unrestricted permutation forces extrapolation: variable importance requires at least one more model, or there is no free variable importance, *Statistics and Computing* 31 (2021).
- [29] R. Tibshirani, Regression shrinkage and selection via the lasso, *Journal of the Royal Statistical Society: Series B (Statistical Methodology)* 58 (1996)
945 267–288.
- [30] M. Stephenson, É. Piette, D. J. N. J. Soemers, C. Browne, Automatic generation of board game manuals, in: C. Browne, A. Kishimoto, J. Schaeffer (Eds.), *Advances in Computers Games (ACG 2021)*, volume 13262 of *Lecture Notes in Computer Science*, Springer, Cham, 2022, pp. 211–222.
- 950 [31] Z. Lin, K.-H. Lam, A. Fern, Contrastive explanations for reinforcement learning via embedded self predictions, in: *Proceedings of the 2021 International Conference on Learning Representations (ICLR)*, 2021.
- [32] H. Baier, M. Kaisers, Explainable search, in: *2020 IJCAI-PRICAI Workshop on Explainable Artificial Intelligence*, 2020.
- 955 [33] H. Baier, M. Kaisers, Towards explainable MCTS, in: *2021 AAAI Workshop on Explainable Agency in AI*, 2021.

- [34] C. R. Silva, M. Bowling, L. H. S. Leles, Teaching people by justifying tree search decisions: An empirical study in curling, *Journal of Artificial Intelligence Research* 72 (2021) 1083–1102.
- 960 [35] A. Pálsson, Y. Björnsson, Evaluating interpretability methods for dnms in game-playing agents, in: C. Browne, A. Kishimoto, J. Schaeffer (Eds.), *Advances in Computer Games (ACG 2021)*, volume 13262 of *Lecture Notes in Computer Science*, Springer, Cham, 2022, pp. 71–81.
- [36] J. Hilton, N. Cammarata, S. Carter, G. Goh, C. Olah, Understanding rl
965 vision, *Distill* (2020). <https://distill.pub/2020/understanding-rl-vision>.
- [37] Y. Coppens, K. Efthymiadis, T. Lenaerts, A. Nowé, Distilling deep reinforcement learning policies in soft decision trees, in: T. Miller, R. Weber, D. Magazzeni (Eds.), *Proceedings of the IJCAI 2019 Workshop on Explainable Artificial Intelligence*, 2019, pp. 1–6.
- 970 [38] Y. Coppens, D. Steckelmacher, C. M. Jonker, A. Nowé, Synthesising reinforcement learning policies through set-valued inductive rule learning, in: F. Heintz, M. Milano, B. O’Sullivan (Eds.), *Trustworthy AI - Integrating Learning, Optimization and Reasoning. TAILOR 2020*, volume 12641 of *Lecture Notes in Computer Science*, Springer, Cham, 2021, pp. 163–179.
- 975 [39] S. Deproost, Neural Tree Distillation to Explain Deep Reinforcement Learning Policies, Masters thesis, Vrije Universiteit Brussel, Brussels, Belgium, 2021.
- [40] G. Liu, O. Schulte, W. Zhu, Q. Li, Toward interpretable deep reinforcement learning with linear model u-trees, in: M. Berlingerio, F. Bonchi,
980 T. Gärtner, N. Hurley, G. Ifrim (Eds.), *Machine Learning and Knowledge Discovery in Databases. ECML PKDD 2018*, *Lecture Notes in Computer Science*, Springer, Cham, 2019, pp. 414–429.
- [41] T. McGrath, A. Kapishnikov, N. Tomašev, A. Pearce, D. Hassabis, B. Kim,

- U. Paquet, V. Kramnik, Acquisition of chess knowledge in AlphaZero,
985 <https://arxiv.org/abs/2111.09259>, 2021.
- [42] R. J. Lorentz, T. E. Zosa IV, Machine learning in the game of Breakthrough, in: M. H. M. Winands, H. van den Herik, W. A. Kusters (Eds.), Advances in Computer Games, volume 10664 of *Lecture Notes in Computer Science*, Springer, 2017, pp. 140–150.
- 990 [43] R. Levinson, R. Snyder, Adaptive pattern-oriented chess, in: Proceedings of the 9th National Conference on Artificial Intelligence, volume 2, AAAI, 1991, pp. 601–605.
- [44] T. Cazenave, Metarules to improve tactical go knowledge, *Information Sciences* 154 (2003) 173–188.
- 995 [45] B. Bouzy, Associating domain-dependent knowledge and monte carlo approaches within a go program, *Information Sciences* 175 (2005) 247–257.
- [46] E. C. D. van der Werf, M. H. M. Winands, H. J. van den Herik, J. W. H. M. Uiterwijk, Learning to predict life and death from Go game records, *Information Sciences* 175 (2005) 258–272.
- 1000 [47] R. Coulom, Computing “Elo ratings” of move patterns in the game of Go, *ICGA Journal* 30 (2007) 198–208.
- [48] D. Silver, R. Sutton, M. Müller, Reinforcement learning of local shape in the game of go, in: Proceedings of the 20th International Joint Conference on Artificial Intelligence, 2007, pp. 1053–1058.
- 1005 [49] T. Raiko, J. Peltonen, Application of UCT search to the connection games of Hex, Y, *Star, and Renkula!, in: Proceedings of the Finnish Artificial Intelligence Conference, 2008, pp. 89–93.
- [50] S.-C. Huang, B. Arneson, R. B. Hayward, M. Müller, J. Pawlewicz, Mohex 2.0: A pattern-based MCTS Hex player, *Computers and Games. CG 2013*
1010 8427 (2014) 60–71.

- [51] C. Browne, É. Piette, M. Stephenson, D. J. N. J. Soemers, General board geometry, in: C. Browne, A. Kishimoto, J. Schaeffer (Eds.), *Advances in Computer Games (ACG 2021)*, volume 13262 of *Lecture Notes in Computer Science*, Springer, Cham, 2022, pp. 235–246.
- 1015 [52] C. Browne, M. Stephenson, É. Piette, D. J. N. J. Soemers, A practical introduction to the Ludii general game system, in: T. Cazenave, H. J. van den Herik, A. Saffidine, I.-C. Wu (Eds.), *Advances in Computer Games. ACG 2019*, volume 12516 of *Lecture Notes in Computer Science (LNCS)*, Springer, Cham, 2020, pp. 167–179.
- 1020 [53] É. Piette, D. J. N. J. Soemers, M. Stephenson, C. F. Sironi, M. H. M. Winands, C. Browne, Ludii – the ludemic general game system, in: *Proceedings of the 24th European Conference on Artificial Intelligence (ECAI 2020)*, volume 325, 2020, pp. 411–418.
- [54] D. Fotland, Knowledge representation in the many faces of Go, <http://www.smart-games.com/knowpap.txt>, 1993.
- 1025
- [55] D. Stern, R. Herbrich, T. Graepel, Bayesian pattern ranking for move prediction in the game of Go, in: W. W. Cohen, A. Moore (Eds.), *Proceedings of the 23rd International Conference on Machine Learning*, 2006, pp. 873–880.
- [56] S. Gelly, D. Silver, Combining online and offline knowledge in UCT, in: *Proceedings of the 24th International Conference on Machine Learning*, 2007, pp. 273–280.
- 1030
- [57] N. Araki, K. Yoshida, Y. Tsuruoka, J. Tsujii, Move prediction in Go with the maximum entropy method, in: *Proceedings of the 2007 IEEE Symposium on Computational Intelligence and Games*, IEEE, 2007, pp. 189–195.
- 1035
- [58] M. Buro, From simple features to sophisticated evaluation functions, in: *First International Conference on Computers and Games*, 1999, pp. 126–145.

- [59] N. R. Sturtevant, A. M. White, Feature construction for reinforcement
1040 learning in Hearts, in: H. J. van den Herik, P. Ciancarini, H. H. L. M. Donkers (Eds.), *Computers and Games*, volume 4630 of *Lecture Notes in Computer Science*, Springer, 2007, pp. 122–134.
- [60] P. Skowronski, Y. Björnsson, M. H. M. Winands, Automated discovery
of search-extension features, in: H. J. van den Herik, P. Spronck (Eds.),
1045 *Advances in Computer Games*, volume 6048 of *Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2009.
- [61] J. Dunn, L. Mingardi, Y. D. Zhuo, Comparing interpretability and explain-
ability for feature selection, <https://arxiv.org/abs/2105.05328>, 2021.
- [62] S. Huang, S. Ontañón, A closer look at invalid action masking in policy
1050 gradient algorithms, <https://arxiv.org/abs/2006.14171>, 2020.
- [63] E. Frank, M. Hall, A simple approach to ordinal classification, in:
L. de Raedt, P. Flach (Eds.), *European Conference on Machine Learning*,
volume 2167 of *Lecture Notes in Computer Science*, Springer, Berlin,
Heidelberg, 2001, pp. 145–156.
- 1055 [64] J. R. Quinlan, Induction of decision trees, *Machine Learning* 1 (1986)
81–106.
- [65] T. Anthony, Z. Tian, D. Barber, Thinking fast and slow with deep learning
and tree search, in: I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fer-
gus, S. Vishwanathan, R. Garnett (Eds.), *Advances in Neural Information*
1060 *Processing Systems* 30, Curran Associates, Inc., 2017, pp. 5360–5370.
- [66] D. J. N. J. Soemers, É. Piette, M. Stephenson, C. Browne, Learning policies
from self-play with policy gradients and mcts value estimates, in: *Proceed-*
ings of the 2019 IEEE Conference on Games, IEEE, 2019, pp. 329–336.
- [67] R. Agarwal, M. Schwarzer, P. S. Castro, A. Courville, M. G. Bellemare,
1065 *Deep reinforcement learning at the edge of the statistical precipice*, in:
Advances in Neural Information Processing Systems, 2021.

- [68] F. Lantz, A. Isaksen, A. Jaffe, A. Nealen, J. Togelius, Depth in strategic games, in: AAAI '17 Workshop on What's Next for AI?, AAAI Press, San Francisco, 2017.
- 1070 [69] D. E. Knuth, R. W. Moore, An analysis of alpha-beta pruning, *Artificial Intelligence* 6 (1975) 293–326.
- [70] M. Ascher, Mu Torere: An analysis of a Maori game, *Mathematics Magazine* 60 (1987) 90–100.
- [71] C. Browne, *Hex Strategy: Making the Right Connections*, AK Peters, Massachusetts, 2000.
- 1075 [72] C. B. Browne, *Connection Games: Variations on a Theme*, AK Peters, Massachusetts, 2005.
- [73] H. Finnsson, Y. Björnsson, Learning simulation control in general game-playing agents, in: *Proceedings of the 24th AAAI Conference on Artificial Intelligence*, AAAI Press, 2010, pp. 954–959.
- 1080 [74] T. Cazenave, Playout policy adaptation with move features, *Theoretical Computer Science* 644 (2016) 43–52.
- [75] J.-T. Saito, M. H. M. Winands, J. W. Uiterwijk, Grouping nodes for Monte-Carlo tree search, in: *Computer Games Workshop, 2007*, pp. 276–283.
- 1085 [76] B. E. Childs, J. H. Brodeur, L. Kocsis, Transpositions and move groups in Monte Carlo tree search, in: *IEEE Symposium on Computational Intelligence and Games, 2008*, pp. 389–395.
- [77] G. van Eyck, M. Müller, Revisiting move groups in Monte-Carlo tree search, in: H. J. van den Herik, A. Plaat (Eds.), *Advances in Computer Games, Lecture Notes in Computer Science*, Springer, Berlin, Heidelberg, 2011, pp. 13–23.
- 1090