Article

Mxplainer: Explain and Learn Insights by Imitating Mahjong Agents

Lingfeng Li [1,†,*], Yunlong Lu [1,], Yongyi Wang [1,], Qifan Zheng [1,] and Wenxin Li [1,]

1 Peking University;
* Correspondence: lingfengli@stu.pku.edu.cn
† Current address: Yiheyuan Road No. 5

Abstract: People need to internalize the skills of AI agents to improve their own capabilities. Our paper focuses on Mahjong, a multiplayer game involving imperfect information and requiring effective long-term decision-making amidst randomness and hidden information. Through the efforts of AI researchers, several impressive Mahjong AI agents have already achieved performance levels comparable to those of professional human players; however, these agents are often treated as black boxes from which few insights can be gleaned. This paper introduces Mxplainer, a parameterized search algorithm that can be converted into an equivalent neural network to learn the parameters of black-box agents. Experiments conducted on AI and human player data demonstrate that the learned parameters provide human-understandable insights into these agents' characteristics and play styles. In addition to analyzing the learned parameters, we also showcase how our search-based framework can locally explain the decision-making processes of black-box agents for most Mahjong game states.

Keywords: Explainable AI; Card Games; Machine Learning

1. Introduction

Games play a pivotal role in the field of AI, offering unique challenges to the research community and serving as fertile ground for the development of novel AI algorithms. Board games, such as Go [1] and chess [2], provide ideal settings for perfect-information scenarios, where all agents are fully aware of the environment states. Card games, like heads-up no-limit hold'em (HUNL) [3,4], Doudizhu [5,6], and Mahjong [7], present different dynamics with their imperfect-information nature, where agents must infer and cope with hidden information from states. Video games, such as Starcraft [8], Minecraft [9], and Honor of Kings [10], push AI algorithms to process and extract crucial features from a multitude of signals amidst noise.

Conversely, the advancement of AI algorithms also incites new enthusiasm in games. The work of AlphaGo [11] in 2016 has had a long-lasting impact on the Go community. It revolutionized the play style of Go, a game with millennia of history, and changed the perspectives of world champions on this game [12]. Teaching tools based on AlphaGo [13] have become invaluable resources for newcomers while empowering professional players to set new records [14].

Mahjong, a worldwide popular game with unique characteristics, has gained traction in the AI research community as a new testbed. It brings its own flavor as a multiplayer imperfect-information environment. First, there is no rank of individual tiles in Mahjong; all tiles are equal in their role within the game. A game of Mahjong is not won by beating

Received:
Revised:
Accepted:
Published:

Citation: Lastname, F.; Lastname, F.; Lastname, F. Mxplainer: Explain and Learn Insights by Imitating Mahjong Agents. *Algorithms* **2025**, *1*, 0. https://doi.org/

other players in card ranks but by being the first to reach a winning pattern. Therefore, state evaluation is more challenging for Mahjong players, as they need to assess the similarity and distance between game states and the closest game goals. Second, the game objective in Mahjong is to become the first to complete one of many possible winning patterns. The optimal goal can frequently change when players draw new tiles during gameplay. In fact, Mahjong's core difficulty lies in selecting the most effective goal among numerous possibilities. Players must evaluate their goals and make decisions upon drawing each tile and reacting to other players' discarded tiles. This decision-making process often involves situations where multiple goals are of similar distance, requiring trade-offs that distinguish play styles and reveal a player's level of expertise.

Several strong agents have already been developed for different variants of Mahjong rules [7,15,16]. Without the use of explainable AI methods[17,18], people can only observe the agents' actions without understanding how the game states are evaluated or which game goals are preferred that lead to those actions.

In Explainable AI (XAI)[19], black-box models typically refer to neural networks that lack inherent transparency and thus rely on post-hoc explanation tools for interpretability [18]. Current post-hoc XAI tools, such as Grad-CAM [20] and LIME [21], are primarily designed for neural networks. While these tools can explain how input features affect outputs, they do not provide insights into agents' decision-making processes.

In this paper, we present Mxplainer (**M**ahjong E**xplainer**), a parameterized classical agent framework designed to serve as an analytical tool for explaining the decision-making process of black-box Mahjong agents. Specifically, we have developed a parameterized framework that forms the basis for a family of search-based Mahjong agents. This framework is then translated into an equivalent neural network model, which can be trained using gradient descent to mimic any black-box Mahjong agent. Finally, the learned parameters are used to populate the parameterized search-based agents. We consider these classical agents to be inherently explainable because each calculation and decision step within them is comprehensible to human experts. This enables detailed interpretation and analysis of the decision-making processes and characteristics of the original black-box agents.

Through a series of experiments on game data from both AI agents and human players, we demonstrate that the learned parameters effectively reflect the decision processes of agents, including their preferred game goals and tiles to play. Our research also shows that by delving into the framework components, we can interpret the decision-making process behind the actions of black-box agents.

This paper pioneers research on analyzing Mahjong agents by presenting Mxplainer, a framework to explain black-box decision-making agents using search-based algorithms. Mxplainer allows AI researchers to profile and compare both AI agents and human players effectively. Additionally, we propose a method to convert any parameterized classical agent into a neural agent for automatic parameter tuning.

2. Related Works

In Explainable AI (XAI)[19] is a research domain dedicated to developing techniques for interpreting AI models for humans. This field encompasses several categories: classification models, generative AI, and decision-making agents. A specific subfield within this domain is Explainable Reinforcement Learning (XRL)[22], which focuses on explaining the behavior of decision-making agents. Explanations in XRL can be classified into two main categories: global and local.

Global explanations provide a high-level perspective on the characteristics and overall strategies of black-box agents, answering questions such as how an agent's strategy differs from others. Local explanations, on the other hand, focus on the detailed decision-making processes of agents, elucidating why an agent selects action A over B under specific scenarios.

XRL methods can be classified into intrinsic and post-hoc methods. Intrinsic methods directly generate explanations from the original black-box models, while post-hoc methods rely on additional models to explain existing ones. Imitation Learning (IL)[23,24] is a family of post-hoc techniques that approximate a target policy. LMUT[25] constructs a U-tree with linear models as leaf nodes and approximates target policies through a node-splitting algorithm and gradient descent. Q-BSP [26] uses batched Q-value to partition nodes and generate trees efficiently. EFS [27] employs an ensemble of linear models with non-linear features generated by genetic programming to approximate target policies. These methods have been tested and excelled in environments such as CartPole, MountainCar, and others in the Gym [28]. However, they rely on properly sampled state-action pairs to generate policies and may not be robust to out-of-distribution (OOD) states, which is particularly crucial for Mahjong with its high-dimensional state space and imperfect information.

PIRL [29] distinguishes itself among IL methods by introducing parameterized policy templates using its proposed policy programming language. It approximates the target π through fitting parameters with Bayesian Optimization in Markov games, achieving high performance in the TORCS car racing game. Compared to TORCS, Mahjong has far more complex state features and requires encoding action histories within states to obtain the Markov property. Additionally, Mahjong agents must make multi-step decisions from game goal selection to tile picking. Similar to PIRL, we define a parameterized search-based framework and optimize parameters using batched gradient descent to address these challenges.

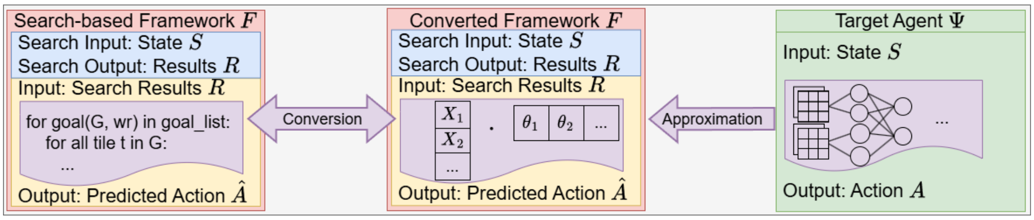


Figure 1. The overview of Mxplainer. Search-based Framework F is a manually-engineered, domain-specific, parameterized template. A component of F can be converted into an equivalent network for supervised learning, where parameters of F serve as neurons. Target agents Ψ are black-box agents to be analyzed, and they can be approximated by the converted F.

3. Methods

We first introduce the general concepts of Mxplainer and then present the details of the implementations in the following subsections. To facilitate readability, we use uppercase symbols to indicate concepts, and lowercase symbols with subscripts for instances of concepts.

Fig. 1 presents the concept overview of Mxplainer. We assume that the search-based nature of F is explainable to experts in the domain, such as Mahjong players and researchers. Within F, there are parameters Θ that control the behaviors of F. To explain a specific target agent Ψ, we would like the behaviors of F to approximate those of Ψ as closely as possible. In order to automate and speed up the approximation process, we convert a part of F that contains Θ into an equivalent neural network representation, and leverage supervised learning to achieve the goals.

F consists of Search Component SC and Calculation Component CC, denoted as $F = SC|CC$. SC searches and proposes valid goals in a fixed order. Next, CC takes groups of manually defined parameters Θ, each of which carries meanings and is explainable to experts, and makes decisions based on the search results from SC, as shown in Fig. 2.

CC can be converted into an equivalent neural network N, whose neurons are semantically equivalent to Θ. $SC|N$ can approximate any target agents Ψ by fitting Ψ's state-action pairs. Since Θ is the same for both CC and N, learned $\hat{\Theta}$ can be put back into CC and explains actions locally through step-by-step deductions of $SC|CC$. Moreover, by normalizing and studying Θ, Mxplainer is able to compare and analyze agents' strategies and characteristics.

The construction of F and the design of parameters Θ are problem-related, as they reflect researchers' and players' attention to the game's characteristics and the game agents' strategies. The conversion from CC to N and the approximation through supervised learning is problem agnostic and can be automated. Out of Mxplainer, SC of F is fixed and the same for all agents, but behaviors of CC and N change as Θ changes for different target agents Ψ.

3.1. Parameters Θ of Framework F

For Θ, we manually craft three groups of parameters to model people's general understanding of Mahjong, Θ_{tile}, Θ_{fan}, and Θ_{meld}.

Θ_{tile} is used to break ties between tiles, and different players may have different preferences for tiles.

Θ_{fan} is designed to break ties between goals when multiple goals are equally distant from the current hand. There are more than billions of possible goals in Mahjong, but each goal consists of **fans**. Thus, we use the compound of preferences of **fans** to sort goals. We hypothesize that there exists a natural order of difficulty between **fans**, which implies that some are more likely to achieve than others. However, such an order is impossible to obtain unless there is an oracle that always gives the best action. Additionally, players break ties based on their inclinations towards **fans**. Since the natural difficulty order and players' inclinations are hard to decouple, we use Θ_{fan} to represent their products.

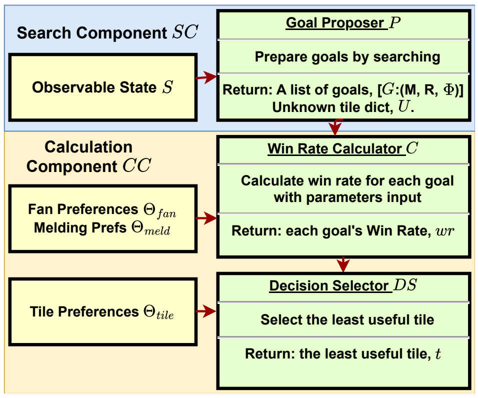


Figure 2. Components of Framework F. Search Component SC uses Goal Proposer P for goal search. Calculation Component CC consists of Win Rate Calculator C and Decision Selector DS, and it is responsible for action calculations. The least useful tile t can be considered as the tile that appears least frequently among the most probable goals. Consequently, removing this tile has the least impact on the expected win rate.

Consequently, Θ_{fan} between different **fans** for a single player cannot be compared directly, but Θ_{fan} for the same **fans** between different players reflect their comparative preferences.

Θ_{meld} are the linear weights of a heuristic function that approximates the probability of **melding** a tile from other players. The linear function takes features from local game states, including the number of total unshown tiles, the length of game steps, and the unshown tile counts of the neighboring tiles. The components and the usage of Θ_{meld} will be discussed in detail in the following sections.

3.2. Search Component SC of Framework F

In Mahjong, **Redundant Tiles** R refer to the tiles in a player's hand that are considered useless for achieving their game goals. In contrast, **Missing Tiles** M are the ones that players need to acquire in future rounds to complete their objectives. Following that, **Shanten Distance** is defined as $D = |M|$, which conveys the distance between the current hand and a selected goal.

Here, we define a game goal as $G = (M, R, \Phi)$, since when both M and R are fixed, a game goal is set, and its corresponding **fans** Φ can be calculated. Each tile $t \in M$ additionally has two indicators, i_p and i_c, which determine if it can be acquired through melding from other players. i_p is true if the player already owns two tiles in the **Pung**, and the same happens to i_c and **Chow**. Thus, $\forall t \in M$, it has (t, i_p, i_c).

Through dynamic programming, we can efficiently propose different combinations of tiles and test if they satisfy MCR's winning condition. We can search all possible goals, but not all goals are reasonable to be considered as candidates. In practice, only up to 64 goals G are returned in ascending **Shanten Distances** D, since in most cases, only the closest goals are important to decisions, and they are usually less than two dozen.

However, only goals are not enough. MCR players also need to consider other observable game information to jointly evaluate the win rate of each goal G, such as other players' discard history and unshown tiles. For our framework F, we only consider unshown tiles U, a dictionary keeping track of the number of each tile that is not shown through observable information.

Thus, the Goal Proposer P accepts game state information S, and outputs $([G], U)$, such that $0 < |[G]| \leq 64$, as shown in Fig. 2-A. In most cases, goals G in $[G]$ can be split into several groups. Different groups contain different **fans** Φ and represent different paths to win, while the goals within each group have different tile combinations for the same **fans** Φ.

3.3. Calculation Component CC of Framework F

Calculation Component CC of Framework F consists of Win Rate Calculator C and Decision Selector DS. CC contains three groups of tunable parameters Θ_{fan}, Θ_{meld}, and Θ_{tile} that control the behavior of Win Rate Calculator C and Decision Selector DS. In all, these three groups of parameters, Θ_{fan}, Θ_{meld}, and Θ_{tile}, are similar to parameters of neural networks, and they are the key factors that control the behaviors of agents derived from framework F.

Table 1. Sizes and meanings of Neural Network N's components. Action **Pung** becomes **Kong** if **Kong** is possible

Entry	Size	Meaning
Overall Output	39	Combined output size
Output (Action)	5	5 Actions: Pass, 3 types of **Chow**, **Pung**
Param Θ_{fan}	80	Preference for all 80 **fans**
Params Θ_{meld}	12	Linear features weights w/ bias
Param Θ_{tile}	34	Preference for all 34 tiles

The Win Rate Calculator C takes the results of the Goal Proposer P, $([G], U)$ as input and estimates the win rate for each goal G. The detailed algorithm of C is shown in Algorithm 1. Simply put, C multiplies the probability of successfully collecting each **missing tile** of a proposed goal as its estimated win rate.

The estimation of win rate in Win Rate Calculator C depends on the two groups of parameters, Θ_{fan} and Θ_{meld}. The probability of acquiring each tile is made of two parts: drawing by oneself or **melding** another player's discarded tile. For each tile t[1], we model the probability of drawing it as $P(t) = U[t]/Z$, in which $Z = sum(U)$. We assume the tile is drawn uniformly from all unshown tiles. The second part of the probability is partly determined by Θ_{meld}, representing agents' optimism of forming **melds**.

As discussed previously, Θ_{meld} is the parameter for a heuristic function that takes in local game features. The features are $\{Z, \frac{1}{Z}, 1 - \frac{1}{Z}, L, \frac{1}{L}, 1\text{-}\frac{1}{L}, U[t-2] : U[t+2], bias\}$, where Z is the number of total unshown tiles, L is the length of game, and $U[t-2] : U[t+2]$ represents the unshown counts of adjacent tiles centered around the tile t. With an additional bias term, Θ_{meld} represents 12 linear weights in total.

For simplicity, uniform distribution is used to estimate the probability of collecting tiles because we designed the learned parameters only to reflect the characteristics of the agents' play styles. On the other hand, tie-breaking between multiple goals with similar **shanten numbers** frequently happens in Mahjong. Players break ties by leaning towards their preferred **fans**, and such preferences are captured by Θ_{fan}. Thus, in Θ_{fan}, a higher value of weight for a specific **fan** represents a higher preference of the agent for this **fan**.

The Decision Selector DS collects results from the Win Rate Calculator C and calculates the final action based on each goal's estimated win rates. The detailed algorithm for S is shown in Algorithm 2. The goal of S is to efficiently select the tile to discard with the most negligible impact on the overall win rate. Heuristically, the required tiles of goals with higher win rates are more important than those with lower ones. Conversely, the **redundant tiles** of such goals are more worthless since their existence actually hinders the goals' completion. Thus, each tile's worthless degree can be computed by accumulating the win rate of goals that regard it as a **redundant tile**, and the tile with the highest worthless degree has the most negligible impact on the overall win rate. Decision Selector DS accepts Θ_{tile} as parameters, which are used similarly as Θ_{fan} to break ties between tiles. For action predictions, such as **Chow** or **Pung**, F records win rates computed by C, assuming those actions are taken, and F selects the action with the highest win rate.

3.4. Differentiable Network N of Mxplainer

The Search-based Framework F is a parameterized search-based agent template, and its parameters Θ need to be tuned to approximate any target agents' behaviors. Luckily, Calculator C and Decision Selector DS only contain fixed-limit loops, with each iteration independent from others and if-else statements whose outcomes can be pre-computed in advance. Thus, C and S can be converted into an equivalent neural network N for parallel computation, and the parameters Θ can be optimized through batched gradient descent. The rules for the conversion can be found in Appendix II.

3.4.1. The Resulting Network N and the Training Objective

The sizes and meanings of the network N outputs and three groups of parameters are reported in Table 1.

The learning objectives depend on the form of target agents Ψ and the problem context. Since we are modeling the selection of actions as classification problems, we use cross entropy (CE) loss between the output of N and the label Y.

[1] t represents a generic tile. $t \in X$ refers to a tile in collection X.

The label can be soft or hard depending on whether the target agent gives probability distributions. Since we cannot access human players' action distribution, we use actions as hard ground-truth labels for all target agents. Additionally, an L2-regularization term of $(\Theta_{fan} - abs(\Theta_{fan}))^2$ is added to penalize negative values of **fan** preferences to make the heuristic parameters more reasonable.

After supervised training, the learned parameters Θ can be directly filled back into CC. Since CC is equivalent to N, $SC|CC(\Theta)$ inherits the characteristics of $SC|N(\Theta)$, approximating the target agent Ψ. By analyzing the parameterized agent $SC|CC(\Theta)$, we can study the parameters to learn the comparative characteristics between agents and gain insights into the deduction process of Ψ through the step-by-step algorithms of Framework $SC|CC$. Since only data pairs are required from the target agents, we can use Mxplainer on both target AI agents and human players.

4. Experiments

We conducted a series of experiments to evaluate the effectiveness of Mxplainer in generating interpretable models and analyze their explainability. Three different target agents are used in these experiments. The first agent ψ_1 is a search-based MCR agent with manually specified characteristics as a baseline. Specifically, this agent only considers the **fan** of *Seven Pairs* to choose, and all its actions are designed to reach this target as efficiently as possible. When multiple tiles are equally good to discard, a fixed order is predefined to break the equality. The second agent ψ_2 is the strongest MCR AI from an online game AI platform, Botzone [30]. The third agent ψ_3 is a human MCR player from an online Mahjong platform, MahjongSoft.

Around 8,000 and 50,000 games of self-play data are generated for the two AI agents. Around 34,000 games of publicly available data are collected from the game website for the single human player over more than 3 years. Through supervised learning on these datasets, three sets of parameters $\theta_{1,2,3}$ are learned and filled back in the Search-based Framework F as-is, making interpretable white-box agents $\hat{\psi}_{1,2,3}$ with similar behavior to the target agents $\psi_{1,2,3}$. The weights for each agent are selected with the highest validation accuracy from three runs. To make weights comparable across different agents, reported weights are normalized with $100 * (\Theta - \Theta_{min}) / \Sigma(\Theta - \Theta_{min})$.

Since it is common in MCR where multiple **redundant tiles** can be discarded in any order, it is difficult to achieve high top-1 accuracy on the validation sets, especially for ψ_2 and ψ_3, which have no preference on the order of tiles to discard. As a reference of the similarity, $\hat{\psi}_{1,2,3}$ achieves top-3 accuracy of 97.15%, 92.75%, 90.12% on the data of $\psi_{1,2,3}$.

4.1. Correlation between Behaviors and Parameters

In the Search-based Framework F, the parameters Θ are designed to be strategy-related features that should take different values for different target agents. Here we analyze the correlation between the learned parameters and target agents' behavior to prove that Mxplainer can extract high-level characteristics of agents and explain their behavior.

Preference of **fans** to choose

Θ_{fan} stands for the relative preference of agents to win by choosing each **fan**. For the baseline target ψ_1 which only chooses the **fan** of *Seven Pairs*, the top values of $\theta_{1,fan}$ learned are shown in Table 2-B. We also include $\theta_{2,fan}$ values for comparisons, demonstrating the differences in learned weights between specialized *Seven Pairs* agent and other agents. It can be seen that the weight for *Seven Pairs* is much higher than those of other **fans**, showing the strong preference of $\hat{\psi}_1$ to this **fan**. The **fans** with the second and third largest weights are patterns similar to *Seven Pairs*, indicating $\hat{\psi}_1$ also tends to approach them during the gameplay based on its learned action choices, though it eventually ends with *Seven Pairs* for 100% of the games. For targets ψ_2 and ψ_3 with unknown preferences of **fans**, we count the frequency of occurrences of each **major fan** in their winning hands as an implication of their preferences and compare these two agents on both the frequency and the learned weights of each **fan**. Table 2-C shows all the **major fans** with a difference of 1% in frequency. Except for the last row, the data shows a significant positive correlation between the preference of target agents ψ and the learned θ_{fan}.

Preference of tiles to discard

Θ_{tile} stands for the relative preference of discarding each tile, especially when multiple **redundant tiles** are valued similarly. Since ψ_2 and ψ_3 show no apparent preference in discarding tiles, we focus on the analysis of ψ_1, which is constructed with a fixed tile preference. We find the learned weights almost form a monotonic sequence with only three

Table 2. A). Defined Order for tiles and their learned weights. B). Sorted weights in the learned parameters $\theta_{1,fan}$ and their corresponding **fans**. Learned weights for $\theta_{2,fan}$ are added for comparison. C). The **major fans** of ψ_2 and ψ_3 with a difference of at least 1% in historical frequency, which is calculated from their game history data.

A	Tile Name	Defined Order	$\theta_{1,tile}$
	White Dragon	1	1.41
	Green Dragon	2	1.13
	Red Dragon	3	1.07
B	**Fan** Name	$\theta_{1,fan}$	$\theta_{2,fan}$
	Seven Pairs	56.19	3.59
	Pure Terminal Chows	7.42	1.99
	Four Pure Shift Pungs	4.10	0.42
C	**Fan** Name	Frequency Diff $\psi_2 - \psi_3$(%)	Param Diff $\theta_{2,fan} - \theta_{3,fan}$
	Pure Straight	1.30	0.45
	Mixed Straight	1.94	0.65
	Mixed Triple Chow	2.09	0.63
	Half Flush	1.04	0.27
	Mixed Shifted Chows	5.08	0.31
	All Types	2.60	0.27
	Melded Hand	-8.25	-1.24
	Fully Concealed Hand	1.83	-0.55

exceptions out of 34 tiles, showing strong correlation between the learned parameters and the tile preferences of the target agent. Table 2-A shows the first a few entries of $\theta_{1,tile}$.

4.2. Manipulation of Behaviors by Parameters

Previous experiments have shown that greater frequencies of **fans** within the game's historical record lead to elevated preferences. In this experiment, we illustrate that through the artificial augmentation of **fan** preferences, the modified agents, in contrast, display elevated frequencies of the corresponding **fans**. We make adjustments to the parameters $\theta_{2,fan}$ within $\hat{\psi}_2$ by multiplying the weight assigned to the *All Types* fan by a factor of 10. This gives rise to the generation of a new agent, ψ_2', which is expected to exhibit a stronger inclination towards selecting this **fan**.

We collect roughly around 8,000 self-play game data samples from ψ_2 and ψ_2'. Subsequently, we determine the frequency with which each fan shows up in their winning hands. The data indicates that among all the fans, only the *All Types* fan undergoes a frequency variation that surpasses 1%. Its frequency has risen from 2.59% to 5.76%, signifying an increment of 3.17%. In contrast, the frequencies of all other **major fans** have experienced changes of less than 1%. Given that Mahjong is a game characterized by a high level of randomness in both the tile-dealing and tile-drawing processes, adjusting the parameters related to fan preferences can only bring about a reasonable alteration in the actual behavior patterns of the target agents. In conjunction with the findings of previous experiments, we can draw the conclusion that the parameters within Mxplainer are, in fact, significantly correlated with the behaviors of agents. Moreover, through the analysis of these parameters, we are capable of discerning the agents' preferences and high-level behavioral characteristics.

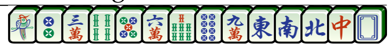
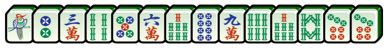
4.3. Interpretation of Deduction Process

In this subsection, we analyze the deduction process of the Search-based Framework F on an example game state by tracing the intermediate results of $\hat{\psi}_2$ to demonstrate the local explainability of Mxplainer agents in decision-making.

The selected game state S is at the beginning of a game with no tiles discarded yet, and the player needs to choose one to discard, as shown in Table 3-A. The target black-box agent ψ_2, selects the tile B9 as the optimal choice to discard for unknown reasons. However, analyses of the execution of the white-box agent $\hat{\psi}_2$ explain the choice.

The Search Component SC proposed 64 possible goals for s, and Table 3-B shows a few goals representative of different **major fans**. With fitted θ_2, Algorithm 1 produces an estimated win rate for each goal G, as listed under the "Win

Table 3. A). An example game state at the beginning of the game where no tiles have been discarded by other players. B). Some proposed goals for state s by $\hat{\psi}_2$ and the estimated win rate. "H&K" is an abbreviation for *Honors and Knitted* due to space constraint.

A		Other Known Information			Current Hand
		None			
B	ID	**Major Fan**	Win Rate	**Redundant Tiles** R	Final Target
	25	*Lesser H&K Tiles*[1]	1.000	D7, C9, B6, B8, B9	
	0	*Knitted Straight*[1]	0.125	C9, B9, NW, RD, WD	
	11	*Pure Straight*	0.018	D2, C3, C6, NW, RD, WD	
	50	Seven Pairs[1]	0.017	B6, B8, B9, NW, RD, WD	

Rate" column of Table 3-B. We observe that though B9 is required in goals such as *Pure Straight*, it is not required for many goals with higher estimated win rates, such as *Knitted Straight*[2].

Following Algorithm 2, we accumulate the win rates for tiles in **Redundant Tiles** R. A higher value of a tile indicates a higher win rate if the tile is discarded, and B9 turns out to have a higher value than other tiles by a large margin, which is consistent with the observed decision of ψ_2.

This section merely demonstrates an analysis of action selection within the context of a simple Mahjong state. However, such analyses can be readily extended to other complex states. The Search Component SC consistently takes in information and puts forward the top 64 reachable goals, ranked according to distances. The Calculation Component CC calculates the win rate for each goal using the learned parameters. Finally, an action is chosen based on these win rates.

Without Mxplainer, people can only observe the actions of black-box MahJong agents, but cannot understand how the decisions are made. Our experiments show that Mxplainer's fitted parameters can well approximate and mimic target agents' behaviors. By examining the fitted parameters and Mxplainer's calculation processes, experts are able to interpret the considerations of black-box agents leading to their actions.

5. Discussion

With Mxplainer, we can compare the differences between agents. By comparing the fitted parameters in parallel with other agents, we can analyze the characteristics of different agents. For example, we can easily observe that the black-box AI agent ψ_2 has a much higher weight on *Thirteen Orphans*, *Seven Pairs*, and *Lesser H&K Tiles*. In contrast, the human player ψ_3 has a significant inclination of making *Melded Hand*, and these observations are indeed backed by their historical wins in Supplemental Material B.

Our proposed approach has a unique advantage in quantifying and tuning weights for custom-defined task-related features in areas where interpretability and performance are crucial. While Mxplainer is specifically designed for Mahjong, we hypothesize that its unique approach and XAI techniques may be applicable to other applications. In fact, we experimentally applied our proposed framework to two examples: Mountain Car from Gym [28] and Blackjack [31]. Both examples, which can be found in Appendix I, confirmed the effectiveness of our proposed method. However, the scope of application of our method still requires further study.

6. Conclusion

In summary, with Mxplainer, we construct, convert, and train domain-specific classical agents with hand-crafted parameters to interpret black-box agents. Our experiments show that Mxplainer can learn meaningful characteristics of Mahjong agents and locally interpret agents' decisions with excellent transparency. Of the three steps in Mxplainer, only the construction of the classical agent is task-related, and the conversion and the training steps can be automated. We hypothesize that this explanation template can be generalized to other problems, and we plan to extend our method for general classical agents and other applications in the future.

[2] A **fan** that do not follow the pattern of four melds and a pair.

Appendix A. Mounter Car and Black Jack

In Mountain Car, the observation space is location **L** and speed **S**, both of which can be positive or negative. The action space are accelerations to the left or right. We can design a heuristic template H which takes the sign of **L** and **S** as input, and the heuristics Θ are actions for these four states. The Transformation T is also simple: we map **L** and **S** to the four states during data processing and output an one-hot mask for Θ. When imitating the optimal strategy, the learned Θ can be found in Figure A1-A, which also achieves optimal control.

In Blackjack, the observation space includes usable ace **A**, dealer's first card **C**, and player's sum of points **S**. The action space is to hit or stick under each state. The Heuristic Template H is designed to act based on heuristic boundaries Θ of **S** for Hit or Stick for each (**A**, **C**) pair. The Transformation T is again mapping (**A**, **C**) pairs to their masks, respectively. The optimal strategy and the learned Θ is shown in Figure A1-B, showing that the framework learns exactly the same behavior with the target model i.e. the optimal strategy.

A.

	L<=0, S<=0	L<=0, S>0	L>0, S<=0	L>0, S>0
Optimal	0	2	0	2
Learned	0.0158	2.0039	-0.0034	2.0047

B.

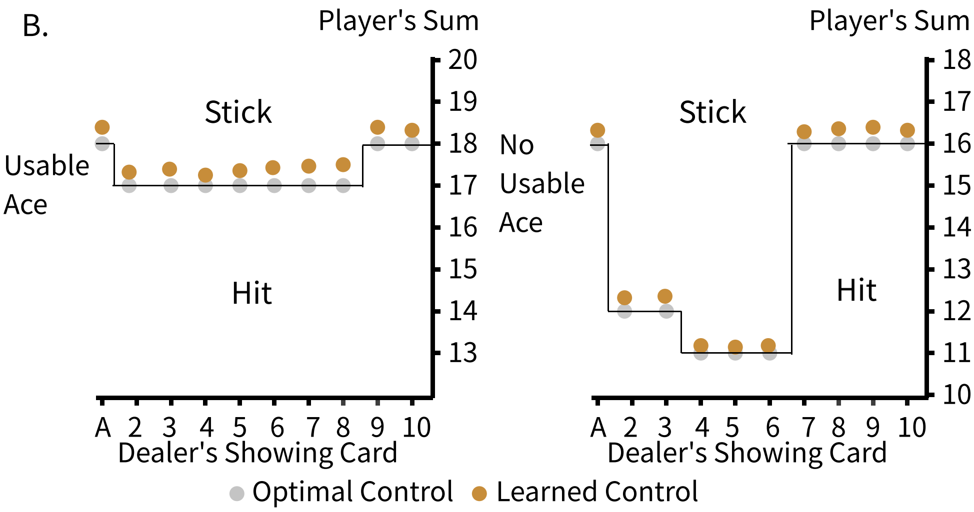


Figure A1. Optimal Parameters and Learned Parameters for A) Mountain Car and B) Black Jack.

Appendix B. Rules and Examples of Transformation of Non-differential Structures

Boolean-to-Arithmetic Masking (M) is defined as a one-hot vector of dim $|n|$ for a conditional statement with n branches. If a conditional branch can be determined without parameters, M can be computed and stored as input data; otherwise, M can be computed at runtime. Computations for results R are the same for all branches, but only one result is activated through $R \cdot M$.

Padding Value (P) is defined as the identity element of its related operator. Identity elements ensure that any computation paths with P produce no-op results, and they are required for the treatment of loops. For example, if the results of a loop are gathered through a summation operation, then the computation path with P produces 0. If the results are gathered through multiplication, then the path with P produces 1. This is because for classical agents to be optimized and parallelized, loops must be set with upper limits, similar to the maximum sequence length when training a recurrent neural network. P needs to fill the iterations where there is no actual data. To keep the Framework F exactly the same as the transformed network N, all upper limits of loops need to be reflected in the original loops of F. Table A1 summarizes M and P.

Table A1. Examples of the transformation for Conditional statements and loops. Note that classical representations of loops need to be modified with the same upper limit as the neural representations. $V^{\top[1,2]}$ is the example input with axis 1 and 2 transposed for efficient representation due to space constraints.

<table>
<tr><th>Conditional</th><th>Classical Representation</th><th>Neural Representation (Pytorch)</th></tr>
<tr><td></td><td>
<pre>
def foo(a,b):
    if P:
        r = a*3+b*4
        return r
    elif Q:
        return a*b
    else
        return a/b
</pre>
</td><td>
<pre>
def forward(A, B, M):
    p = A*3 + B*4
    q = A*B
    o = A/B
    out = torch.cat([p,q,o], dim=1)
    return torch.sum(out*M, dim=1)
</pre>
</td></tr>
<tr><th>Loop A</th><th>Classical Representation</th><th>Neural Representation (Pytorch)</th></tr>
<tr><td>Note: Results are independent between Iterations.</td><td>
<pre>
def bar1(A):
    res = []
    e = enumerate
    for i,a in e(A):
        if i>64:
            break
        r = 2*(a+1)
        res.append(r)
    p = np.prod
    return p(res)
</pre>
</td><td>
<pre>
def forward(V):
    V1 = V.view(n*64, 2)
    O = V1[:,0]*2*(V1[:,1]+1)
    # convert padding values to 1
    # for the final torch.prod()
    O += 1-V1[:, 0]
    O1 = O.view(n, 64)
    return torch.prod(O1, dim=1)
</pre>
</td></tr>
<tr><th>Loop B</th><th>Classical Representation</th><th>Neural Representation (Pytorch)</th></tr>
<tr><td>Note: Results are dependent between Iterations.</td><td>
<pre>
def bar2(A):
    res = []
    e = enumerate
    for i,a in e(A):
        if i>64:
            break
        r=a
        if i!=0:
            r += res[-1]
        res.append(r)
    return sum(res)
</pre>
</td><td>
<pre>
def forward(V):
    res = []
    for i in range(64):
        if i==0:
            r = V[:, i, 1]
        else:
            # computing real data
            r = res[-1]+V[:, i, 1]
        # set zero for padding values
        r*=V[:, i, 0]
        res.append(r)
    res = torch.stack(res)
    res = torch.sum(res, dim=0)
    return res
</pre>
</td></tr>
</table>

Appendix C. Mahjong the Game

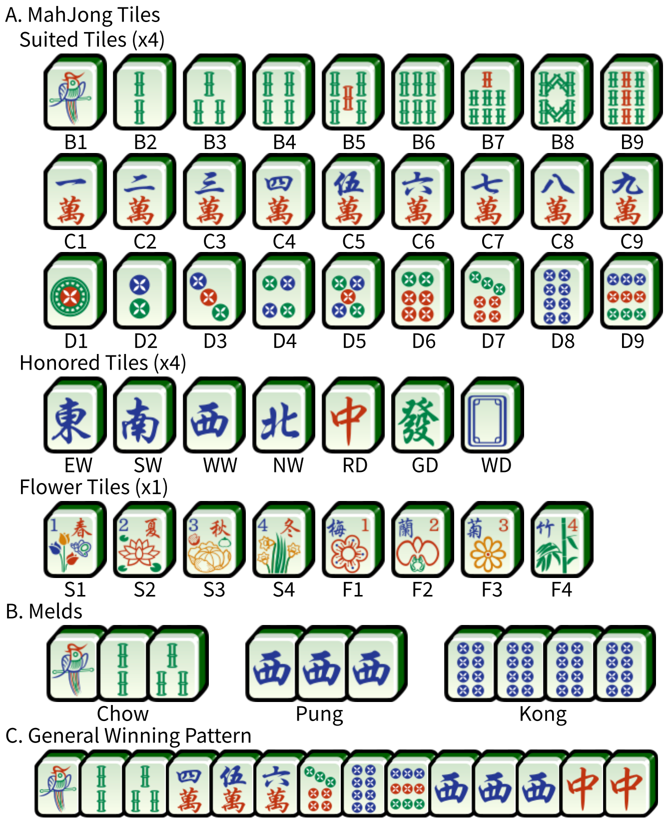


Figure A2. Basics of Mahjong. A). All the Mahjong tiles. There are four identical copies for suited tiles and honored tiles, and one copy for each flower tile. B). Examples of Chow, Pung, and Kong. Note that only suited tiles are available for Chow. C). Example of the general winning pattern.

Mahjong is a four-player imperfect information tile-based tabletop game. The complexity of imperfect-information games can be measured by information sets, which are game states that players cannot distinguish from their own observations. The average size of information sets in Mahjong is around 10^{48}, making it a much more complex game to solve than Heads-Up Texas Hold'em [32], whose average size of information sets is around 10^3. To facilitate the readability of this paper, we highlight terminologies used in Mahjong with **bold texts**, and we distinguish scoring patterns (**fans**) by *italicized texts*.

In Mahjong, there are a maximum of 144 tiles, as shown in Fig. A2-A. Despite its plethora of rule variants, Mahjong's general rules are the same. On a broad level, Mahjong is a pattern-matching game. Each player begins with 13 tiles only observable to themselves, and they take turns to draw and discard one tile until one completes a game goal with a 14th tile. The general pattern of 14 tiles is four **melds** and a pair, as shown in Fig. A2-C. A meld can take the form of **Chow**, **Pung**, and **Kong**, as shown in Fig. A2-B. Apart from drawing all the tiles by themselves, players can take the tile just discarded by another player instead of drawing one to form a **meld**, called **melding**, or declare a win.

Appendix C.1. Official International Mahjong

Official International Mahjong stipulates Mahjong Competition Rules (MCR) to enhance the game's complexity and competitiveness and weaken its gambling nature. It specifies 80 scoring patterns with different points, called "**fan**", ranging from 1 to 88 points. In addition to the winning patterns of four melds and a pair, players must have at least 8 points by matching multiple scoring patterns to declare a win.

Specific rules and requirements for each pattern can be found in this book [33]. Of 81 **fans**, 56 are highly valued and called **major fans** since most winning hands usually consist of at least one. The standard strategy of MCR players is to make game plans by identifying several **major fans** closest to their initial set of tiles. Then, depending on their incoming

Algorithm 1 Win Rate Estimation for A Single Goal

Input: Goal G: $<$ **missing tiles** M:$<t$, **Chow** indicator i_c, **Pung** indicator $i_p$$>$, **fan** list F $>$, unshown dict U, game length L
Parameter: Θ_{fan}, Θ_{meld}
Output: Estimated win rate for goal G

Initialize win rate $wr \leftarrow 100$
for all missing tile $m \in M$ **do**
  // Construct local tile feature x_m from game length L,
  // and remaining adjacent tile count U.
  $x_m \leftarrow U, L$
  // Calculate prob. of drawing m and others discarding m
  $p_{draw} \leftarrow U[m]/sum(U)$
  $p_{discard} \leftarrow U[m]/sum(U) * \Theta_{meld} * x_m$
  $source \leftarrow 0$
  if i_p **then**
    $source \leftarrow 3$ // one can pung from all others
  else if i_c **then**
    $source \leftarrow 1$ // one can only **chow** from the one left
  end if
  $p_{meld} \leftarrow p_{discard} * source$
  $wr \leftarrow wr \times (p_{draw} + p_{meld})$
end for
Total **fan** weight $fw \leftarrow \sum \Theta_{fan}[f]$ for **fan** $f \in F$
$wr \leftarrow wr \times fw$
return wr

Algorithm 2 Discarding Tile Selection

Input: List of goals and their win rates $L = [(\text{Goal } G, \text{Win rate } wr)]$, Hand tiles H
Parameter: Θ_{tile}
Output: Tile to discard

Initialize tile values $d \leftarrow$ Dict $\{\text{tile } t : 0\}$
for all $(G, wr) \in L$ **do**
  for all tile $t \in G$ **do**
    $d[t] \leftarrow d[t] + wr$ // **redundant tiles**
  end for
end for
for all tile $t \in d$ **do**
  $d[t] \leftarrow d[t] \times \Theta_{tile}[t]$
end for
return $\arg\max_{t \in H} d[t]$

tiles, they gradually select one of them as the terminal goal and strive to collect all the remaining tiles before others do. The exact rules of MCR are detailed in *Official International Mahjong: A New Playground for AI Research* [32].

D. Algorithms

References

1. Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. Mastering the game of Go without human knowledge. *Nature* **2017**, *550*, 354–359. https://doi.org/10.1038/nature24270.
2. Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. Mastering Chess and Shogi by Self-Play with a General Reinforcement Learning Algorithm, 2017, [arXiv:cs.AI/1712.01815].
3. Brown, N.; Sandholm, T. Superhuman AI for heads-up no-limit poker: Libratus beats top professionals. *Science* **2018**, *359*, 418–424, [https://www.science.org/doi/pdf/10.1126/science.aao1733]. https://doi.org/10.1126/science.aao1733.
4. Brown, N.; Sandholm, T. Superhuman AI for multiplayer poker. *Science* **2019**, *365*, 885–890, [https://www.science.org/doi/pdf/10.1126/sci https://doi.org/10.1126/science.aay2400.

5. Yang, G.; Liu, M.; Hong, W.; Zhang, W.; Fang, F.; Zeng, G.; Lin, Y. PerfectDou: Dominating DouDizhu with Perfect Information Distillation, 2024, [arXiv:cs.AI/2203.16406].
6. Zha, D.; Xie, J.; Ma, W.; Zhang, S.; Lian, X.; Hu, X.; Liu, J. DouZero: Mastering DouDizhu with Self-Play Deep Reinforcement Learning, 2021, [arXiv:cs.AI/2106.06135].
7. Li, J.; Koyamada, S.; Ye, Q.; Liu, G.; Wang, C.; Yang, R.; Zhao, L.; Qin, T.; Liu, T.Y.; Hon, H.W. Suphx: Mastering Mahjong with Deep Reinforcement Learning, 2020, [arXiv:cs.AI/2003.13590].
8. Vinyals, O.; Babuschkin, I.; Czarnecki, W.M.; Mathieu, M.; Dudzik, A.; Chung, J.; Choi, D.H.; Powell, R.; Ewalds, T.; Georgiev, P.; et al. Grandmaster level in StarCraft II using multi-agent reinforcement learning. *Nature* **2019**, *575*, 350–354. https://doi.org/10.1038/s41586-019-1724-z.
9. Fan, L.; Wang, G.; Jiang, Y.; Mandlekar, A.; Yang, Y.; Zhu, H.; Tang, A.; Huang, D.A.; Zhu, Y.; Anandkumar, A. MineDojo: Building Open-Ended Embodied Agents with Internet-Scale Knowledge, 2022, [arXiv:cs.LG/2206.08853].
10. Wei, H.; Chen, J.; Ji, X.; Qin, H.; Deng, M.; Li, S.; Wang, L.; Zhang, W.; Yu, Y.; Liu, L.; et al. Honor of Kings Arena: an Environment for Generalization in Competitive Reinforcement Learning, 2022, [arXiv:cs.LG/2209.08483].
11. Silver, D.; Huang, A.; Maddison, C.J.; Guez, A.; Sifre, L.; van den Driessche, G.; Schrittwieser, J.; Antonoglou, I.; Panneershelvam, V.; Lanctot, M.; et al. Mastering the game of Go with deep neural networks and tree search. *Nature* **2016**, *529*, 484–489. https://doi.org/10.1038/nature16961.
12. Willingham, E. AI's Victories in Go Inspire Better Human Game Playing. https://www.scientificamerican.com/article/ais-victories-in-go-inspire-better-human-game-playing/, 2023. Accessed: 2024-4-12.
13. Huang, A.; Hui, F.; Baker, L. AlphaGo Teach. https://alphagoteach.deepmind.com/, 2017. Accessed: 2024-4-12.
14. Saedol, L. 8 years later: A world Go champion's reflections on AlphaGo. https://blog.google/around-the-globe/google-asia/8-years-later-a-world-go-champions-reflections-on-alphago/, 2024. Accessed: 2024-4-12.
15. Li, J.; Wu, S.; Fu, H.; Fu, Q.; Zhao, E.; Xing, J. Speedup Training Artificial Intelligence for Mahjong via Reward Variance Reduction. In Proceedings of the 2022 IEEE Conference on Games (CoG). IEEE Press, 2022, p. 345–352. https://doi.org/10.1109/CoG51982.2022.9893584.
16. Zhao, X.; Holden, S.B. Building a 3-Player Mahjong AI using Deep Reinforcement Learning, 2022, [arXiv:cs.AI/2202.12847].
17. Arrieta, A.B.; Díaz-Rodríguez, N.; Ser, J.D.; Bennetot, A.; Tabik, S.; Barbado, A.; García, S.; Gil-López, S.; Molina, D.; Benjamins, R.; et al. Explainable Artificial Intelligence (XAI): Concepts, Taxonomies, Opportunities and Challenges toward Responsible AI, 2019, [arXiv:cs.AI/1910.10045].
18. Linardatos, P.; Papastefanopoulos, V.; Kotsiantis, S. Explainable AI: A Review of Machine Learning Interpretability Methods. *Entropy* **2021**, *23*. https://doi.org/10.3390/e23010018.
19. Barredo Arrieta, A.; Díaz-Rodríguez, N.; Del Ser, J.; Bennetot, A.; Tabik, S.; Barbado, A.; Garcia, S.; Gil-Lopez, S.; Molina, D.; Benjamins, R.; et al. Explainable Artificial Intelligence (XAI): Concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion* **2020**, *58*, 82–115. https://doi.org/https://doi.org/10.1016/j.inffus.2019.12.012.
20. Selvaraju, R.R.; Das, A.; Vedantam, R.; Cogswell, M.; Parikh, D.; Batra, D. Grad-CAM: Why did you say that? Visual Explanations from Deep Networks via Gradient-based Localization. *CoRR* **2016**, *abs/1610.02391*, [1610.02391].
21. Ribeiro, M.T.; Singh, S.; Guestrin, C. "Why Should I Trust You?": Explaining the Predictions of Any Classifier. *CoRR* **2016**, *abs/1602.04938*, [1602.04938].
22. Milani, S.; Topin, N.; Veloso, M.; Fang, F. Explainable Reinforcement Learning: A Survey and Comparative Review. *ACM Comput. Surv.* **2024**, *56*. https://doi.org/10.1145/3616864.
23. Abbeel, P.; Ng, A.Y. Apprenticeship learning via inverse reinforcement learning. In Proceedings of the Proceedings of the Twenty-First International Conference on Machine Learning, New York, NY, USA, 2004; ICML '04, p. 1. https://doi.org/10.1145/1015330.1015430.
24. Schaal, S. Is imitation learning the route to humanoid robots? *Trends in Cognitive Sciences* **1999**, *3*, 233–242. https://doi.org/https://doi.org/10.1016/S1364-6613(99)01327-3.
25. Liu, G.; Schulte, O.; Zhu, W.; Li, Q. Toward Interpretable Deep Reinforcement Learning with Linear Model U-Trees. In Proceedings of the Machine Learning and Knowledge Discovery in Databases; Berlingerio, M.; Bonchi, F.; Gärtner, T.; Hurley, N.; Ifrim, G., Eds., Cham, 2019; pp. 414–429.
26. Jhunjhunwala, A.; Lee, J.; Sedwards, S.; Abdelzad, V.; Czarnecki, K. Improved Policy Extraction via Online Q-Value Distillation. In Proceedings of the 2020 International Joint Conference on Neural Networks (IJCNN), 2020, pp. 1–8. https://doi.org/10.1109/IJCNN48605.2020.9207648.
27. Zhang, H.; Zhou, A.; Lin, X. Interpretable policy derivation for reinforcement learning based on evolutionary feature synthesis. *Complex & Intelligent Systems* **2020**, *6*, 741–753. https://doi.org/10.1007/s40747-020-00175-y.
28. Brockman, G.; Cheung, V.; Pettersson, L.; Schneider, J.; Schulman, J.; Tang, J.; Zaremba, W. OpenAI Gym, 2016, [arXiv:cs.LG/1606.01540].

29. Verma, A.; Murali, V.; Singh, R.; Kohli, P.; Chaudhuri, S. Programmatically Interpretable Reinforcement Learning. In Proceedings of the Proceedings of the 35th International Conference on Machine Learning; Dy, J.; Krause, A., Eds. PMLR, 10–15 Jul 2018, Vol. 80, *Proceedings of Machine Learning Research*, pp. 5045–5054.
30. Zhou, H.; Zhang, H.; Zhou, Y.; Wang, X.; Li, W. Botzone: An Online Multi-Agent Competitive Platform for AI Education. In Proceedings of the Proceedings of the 23rd Annual ACM Conference on Innovation and Technology in Computer Science Education, New York, NY, USA, 2018; ITiCSE 2018, p. 33–38. https://doi.org/10.1145/3197091.3197099.
31. Sutton, R.S.; Barto, A.G. *Reinforcement Learning: An Introduction*; A Bradford Book: Cambridge, MA, USA, 2018.
32. Lu, Y.; Li, W.; Li, W. Official International Mahjong: A New Playground for AI Research. *Algorithms* **2023**, *16*. https://doi.org/10.3390/a16050235.
33. Novikov, V. Handbook on Mahjong Competition Rules, 2016. Accessed: 2023-07-17.