electronics

MDPI

Article

Imperfect-Information Game AI Agent Based on Reinforcement Learning Using Tree Search and a Deep Neural Network

Xin Ouyang * and Ting Zhou

School of Information Engineering and Automation, Kunming University of Science and Technology, Kunming 650500, China; z1749904953@gmail.com
* Correspondence: kust19843456@gmail.com

Abstract: In the field of computer intelligence, it has always been a challenge to construct an agent model that can be adapted to various complex tasks. In recent years, based on the planning algorithm of Monte Carlo tree search (MCTS), a new idea has been proposed to solve the AI problems of two-player zero-sum games such as chess and Go. However, most of the games in the real environment rely on imperfect information, so it is impossible to directly use the normal tree search planning algorithm to construct a decision-making model. Mahjong, which is a popular multiplayer game with a long history in China, attracts great attention from AI researchers because it contains a large game state space and a lot of hidden information. In this paper, we utilize an agent learning approach that leverages deep learning, reinforcement learning, and dropout learning techniques to implement a Mahjong AI game agent. First, we improve the state transition of the tree search based on the learned MDP model, the player position variable and transition information are introduced into the tree search algorithm to construct a multiplayer search tree. Then, the model training based on a deep reinforcement learning method ensures the stable and sustainable training process of the learned MDP model. Finally, we utilize the strategy data generated by the tree search and use the dropout learning method to train the normal decision-making agent. The experimental results demonstrate the efficiency and stability performance of the agent trained by our proposed method compared with existing agents in terms of test data accuracy, tournament ranking performance, and online match performance. The agent plays against human players and acts like real humans.

Keywords: artificial intelligence; game agent; reinforcement learning; machine learning; neural network; tree search

Citation: Ouyang, X.; Zhou, T. Imperfect-Information Game AI Agent Based on Reinforcement Learning Using Tree Search and a Deep Neural Network. *Electronics* **2023**, *12*, 2453. https://doi.org/10.3390/electronics12112453

Academic Editors: Yu-Chen Hu, Praveen Kumar Donta, Piyush Kumar Pareek and Chinmaya Kumar Dehury

Received: 17 April 2023
Revised: 23 May 2023
Accepted: 24 May 2023
Published: 29 May 2023

1. Introduction

Since the advent of computer games, game AI [1] has been one of the most important branches of research in AI [2]. In recent decades, game AI has made significant progress in games with perfect information, such as AlphaGo [3] and AlphaZero [4], through deep convolutional networks [5] and tree search techniques. In contrast, related research on games with imperfect information [6] has not had such outstanding success. Currently, some imperfect-information games are solved with CFR [7] or reinforcement learning methods [8,9]. However, due to the complexity of the environment rules and the high simulation cost, both models are inefficient and unstable.

In the game field of imperfect information, the research and development are completely different from that of perfect information, and it can be said that the development is very slow. The main reason is that, in an imperfect-information environment, participants or agents usually cannot fully access all the information about the current situation. It is difficult to utilize search algorithms to improve agent strategies. Furthermore, the amount of state information that needs to be processed is huge. In most cases, participants or agents can only observe private information and partly public features, and the key information

that has a significant impact on the agent's strategies is usually hidden. As a result, the traditional search-based learning methods cannot be directly applied to imperfect-information game scenarios.

The intelligent agent-based modeling approach has become popular in recent studies, because it can provide a direct and specific way to overcome complex real-world problems. With the help of machine learning (ML) and reinforcement learning (RL) techniques, an opportunity to build and improve an agent-based model has arrived. Shahbazi et al. [10] proposed an intelligent agent-based recommendation system based on NLP techniques and semantic analysis approaches and achieved state-of-the-art prediction accuracy. Alejandro et al. [11] proposed an extension to the design concepts and details (ODD) protocol to support agent-based modeling and offered a standardized description of ML approaches. Furthermore, some other researchers attempted to integrate RL with a neural network. Arash Heidari et al. [12] combined the lightweight version of the RL technique and a convolutional neural network to successfully provide a good performance of an offloading strategy. Other researchers have provided a review of the literature: Zahra Amiri et al. [13] reviewed and reported a well-organized classification of distributed system, and divided several recent studies into seven divisions to analyze their advantages and drawbacks. Currently, most agent-based modeling approaches are based on a certain strategy or pure supervised learning methods, which is ineffective and provides poor performance.

In this paper, we design an agent model of Mahjong based on reinforcement learning, deep learning, and dropout learning to overcome some of the mentioned limitations. The detailed rules of Mahjong can be found in Appendix A. However, implementing and training an agent model directly from the simulation of imperfect-information games with complex rules, such as Mahjong, presents a huge challenge. Without a carefully designed agent model and an effective learning approach, the agent may not be able to effectively learn the game strategies for good performance. To tackle the problem of the huge hidden information and large state-action space, in this paper, we design an extended multiplayer game search tree in a way that allows for more flexible state transitions and dynamic decision making. After that, by using dropout learning based on a dynamic Bernoulli random matrix, we transform it into a normal agent with imperfect information to extend the applicability of the agent. The contributions of this article are as follows:

1. An extended multiplayer game search tree combined with a reinforcement learning (RL) technique and deep learning is proposed. We incorporate an internalized MDP network model to internalize the simulation state and reduce the simulation cost. Then, we also introduce the additional player position and state-action information into the search tree and network policy estimation to realize the dynamic transition of the internal state.
2. A normal agent with imperfect information is trained through dropout learning. With the utilization of the improved strategies achieved by the RL agent and the dropout learning technique, our approaches can effectively handle the decision-making problems of Mahjong.
3. To increase the learning efficiency of the proposed agent, residual blocks [5], an experience replay buffer [14], and an L2 regularization term [15,16] are used. In addition, extra noise and a multi-thread simulation are applied to the search tree to improve the performance of the search algorithm.
4. The convergence and performance of our method are demonstrated by successfully implementing a Mahjong agent with good performance and comparing it to several existing models through accuracy on test data and tournament confrontation. Finally, we conduct tests on online game platforms against human players to further validate the practicality of the proposed agent.

The structure of this paper is as follows: Section 2 reviews the related work, including both model-free and model-based agent learning approaches. In Section 3, we describe the game scenario of Mahjong and detail the framework of the RL methodology. In Section 4, we report the simulation and experiment results, and compare them with previous work to

demonstrate the performance of our agent. Finally, in Section 5, we conclude this article and discuss improvement works in the future.

2. Related Work

At present, reinforcement learning methods generally have two principal categories: model-based, and model-free [17]. It is very popular among researchers to utilize model-free methods, such as policy gradient [18] and advantage function [19], to directly implement decision-making agents. The model-free methods are usually simple and direct, and can quickly train an agent with good performance for most uncomplicated games.

However, when interacting with complex rule environments, the model-free RL method encounters problems such as a low convergence speed and large gradient fluctuations during the training, e.g., TD learning [20] and Q learning [21]. In addition, additional labeling of training data is usually required for the agent to speed up model training. In contrast, the model-based RL method first constructs a specific environment model, generally based on the Markov decision process (MDP) [22]. Then, it is combined with the tree search planning and deep learning techniques to handle more complex decision-making problems, such as the TreeQN [23] model. However, the model-based methods usually need to solve two key issues additionally. First, the tree search algorithm usually not only needs to implement a simulation accurately but also needs to maintain and restore all the simulation states, which is highly reliant on the knowledge of the environment, e.g., an accurate environmental simulator; secondly, it cannot be directly applied to imperfect-information environments, because a large amount of hidden information could weaken the performance of the search algorithm. As a result, an agent could only obtain a suboptimal strategy.

Aiming at the first problem, Julian Schrittwieser et al. [24] proposed a learned model that uses the locally learned MDP model to replace the accurate environmental simulator. The model introduces the concept of internal state and learns and fits the state transition of tree search simulations by using the deep-learning-based MDP model that is iteratively updated. The MDP model is viewed as the hidden layers of deep neural networks, and the internal state is used to approximate the optimal policy function and value function. However, it is only suitable for single-player or two-player games with simple rules and it cannot be directly applied to the mostly imperfect-information games.

Imperfect-information agents are generally difficult to implement because observers or participants usually only observe part of the information. To solve this problem, some researchers enumerated all possible states and used determinization technology to achieve improved strategies [25]. Since all possible results also need to be averaged, this method may also obtain a suboptimal strategy [26]. As a result, it does not find the global optimal solution but a local solution. Different from the above approaches, some researchers have proposed a learning agent based on the dropout technique [27], which trains the agent from scratch and slowly drops part of the global features during RL training, and realizes the conversion from a perfect information agent to a normal agent with imperfect information. However, its training method has a slow convergence and low effectiveness. Essentially, it is based on the model-free approach. Not only does this strategy lack optimality but also the learning process is prone to fluctuations, making it hard to optimize the agent model.

Mahjong is one of the most complex imperfect-information games. Players or agents need to take action based on their hand's tiles and other players' public tiles. It is very difficult to directly apply tree search algorithms to Mahjong because the game state and state transitions are complicated. Tsuruoka et al. [28] considered the opponents' possible moves or actions and proposed a Mahjong agent combined with the opponent model. Van Rijn et al. [29] constructed a Mahjong search tree based on statistical methods and machine learning approaches to find the action with the maximum possible return. Another researcher proposed a strategy model based on deep learning [30] and deep convolutional networks. Researchers from Microsoft Asia proposed Suphx [31], based on distributed

reinforcement learning combined with other novel improvement methods, which defeated professional players on the Riichi Mahjong platform.

In recent years, Gao et al. [32] used DenseNet [33] to extract all the features of Mahjong and obtained a better strategy with the XGBoost [34] algorithm, which achieved great accuracy on the test dataset. Zheng et al. [35] described recent Mahjong AI works in detail, summarizing the current research, analyzing the differences of each approach, and proposing the future work of Mahjong AI.

In conclusion, the studies reviewed above emphasize the potential of RL approaches in designing agents for imperfect-information games that could solve decision-making problems and outperform human players.

3. Reinforcement Learning Algorithm

3.1. Tree Search Learning Algorithm for Mahjong

3.1.1. Internalized MDP-Based MCTS Learning Algorithm

Due to the complexity of the game rules of Mahjong, it is difficult to implement the search simulations with accurate Mahjong simulators. For example, not only should the game state of the Mahjong be maintained, but also the backup and restore operations of the simulators should be performed. This is to ensure the accuracy of the game state and state transition. To reduce the dependency on accurate simulators, we use the internalized MDP model-based RL method in combination with the multiplayer tree search algorithm to improve and obtain the agent strategy. In this work, the internalized MDP model was used instead of the accurate Mahjong simulator, combined with the MCTS algorithm to decrease the execution cost of simulators and further strengthen the Mahjong game strategy. Note that the tree search process can access all the feature information of Mahjong to speed up the agent learning process and ensure the accuracy and efficiency of the tree search.

The internalized MDP model extracts the characteristic information of Mahjong based on a deep convolutional network [36] and then relies on the information processing and integration capabilities of a recurrent neural network [37] to further improve the generalization and estimation capabilities of the functions to obtain more accurate value estimation and prior policy estimation of the Mahjong situation. It is also combined with the MCTS algorithm to strengthen the agent strategy. The internalized MDP model consists of three sub-networks: the representation network is used to calculate the network state of the root node of the search tree; the dynamic network is used to calculate the internal state of the next node during the simulation; the prediction network is used to calculate the prior policy estimation and the multi-value estimation corresponding to the player state of the current leaf node. We utilize the residual blocks as the main body of the representation and dynamic network to extract the features of the Mahjong situation. The information on the Mahjong situation is represented as the internal state of the network during the search process. Now, we describe the internalized MDP-based MCTS algorithm for Mahjong. In addition to the original search algorithm, we also introduce the player position variable into the search algorithm, which can handle dynamic player priority and makes dynamic decision-making possible. Note that the search algorithm also conforms to the principle of the Minimax algorithm. To simplify, the player position variables are represented in relative order and start from 0. In addition, the player's basic action is combined with additional dynamic information to let the search tree adapt to the dynamic Mahjong game rules. Formally, at each step t, the model performs a strategy search process to determine the current player's best action. A search process consists of N multiple simulations and each simulation involves several steps to calculate and predict k $(k = 0, 1, l)$ hypothetical actions in the future. After the N simulations are implemented completely, the agent will obtain the strategy probability vector corresponding to the number of visit counts about all edges of the root node of the search tree. The tree search process and training process are shown in Figure 1.

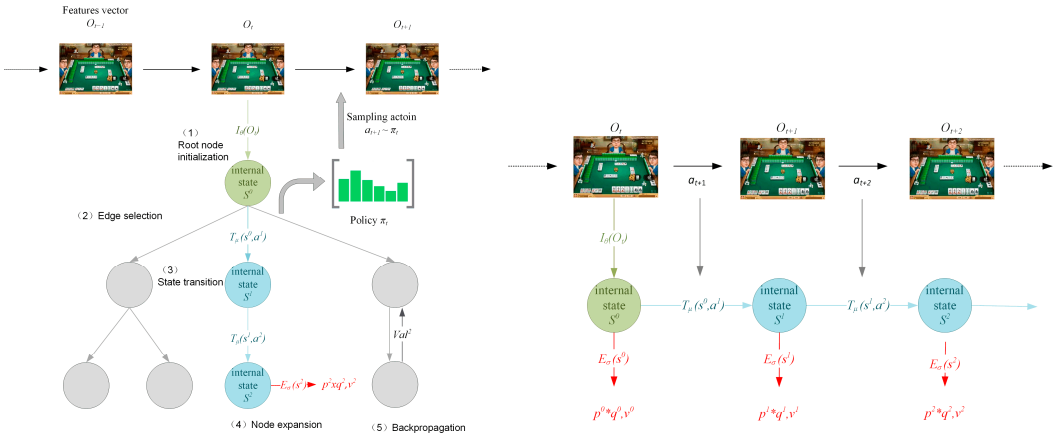


Figure 1. The internalized MDP-based MCTS search process (**left**) and the MDP network training process (**right**). The search process mainly consists of five parts, mentioned below. The network training process is very similar to the search process, but limited to a fixed number of steps.

To better describe the steps of each simulation, we first declare the variables used in the MCTS algorithm [38]. Each node stores its game state, represented as the internal network state S, and the information associated with all edges, such as the number of visits N, the average state-action value Q, the player position variable D, and the prior policy estimation P. Then, we perform the following five steps sequentially, starting from the root node for each simulation:

1. Initialization of the root node. If the root node of the Mahjong game tree has not been initialized before, we use the representation network to calculate its internal state:

$$S^0 = I_\theta(O_t) \tag{1}$$

where $I_\theta(O_t)$ denotes the representation network parameterized by θ with the features vector O_t of Mahjong as the input. Then, skip to step 4. Otherwise, go to step 2.

2. Selection of edges. Selecting the edge along a path with the upper confidence bound (UCB) algorithm [39] until an uninitialized (empty) leaf node is reached. In detail, the statistics of the current node corresponding to the current player are used to determine the next search edge. The UCB algorithm used is shown below:

$$a^k = argmax_a \left[Q(s,a) + C_1 \frac{\sqrt{\Sigma_b N(s,b)}}{1+N(s,a)} P(s,a) \right] \tag{2}$$

In Equation (2), the key point is that the exploitation and exploration of the search depend on the value Q and the policy estimation P. It can be seen that the parameter $\frac{\sqrt{\Sigma_b N(s,b)}}{1+N(s,a)}$ is related to the edge visits $N(s,a)$ and the parent node visits $\Sigma_b N(s,b)$.

The smaller the edge visits, the higher the priority of the edge, and the parent node visits provide an additional exploration for each child edge. The variable C_1 controls the overall exploration of the search. Due to the large action and state space of Mahjong, $C_1 = 1.5$ in our experiment. The purpose is to encourage more exploration.

3. State transition. After reaching an uninitialized leaf node, we utilize the dynamic network to compute the internal state s^k of this node with its parent internal state (the latest state) s^{k-1} and the transition action a^k combined with additional Mahjong transition information as the network input:

$$s^k = T_\mu\left(s^{k-1}, a^k\right) \tag{3}$$

where T_μ denotes the dynamic network parameterized by μ. After that, the resulting internal state s^k will be used to compute related estimation variables of the leaf node in step 4.

4. Initialization of the leaf node. After obtaining the internal state $s^k (k = 0, 1, \ldots)$ of the new leaf node, we need to create and initialize all edges of this node. First, we use the prediction network to calculate the prior policy p^k in combination with rule-related transition information q^k of Mahjong and the value estimation vector v^k of all players:

$$p^k \times q^k, v^k = E_\sigma\left(s^k\right) \tag{4}$$

where E_σ denotes the prediction network parameterized by σ. Then, we initialize all edges of this node, e.g., $N(s, a) = 0, Q(s, a) = 0$, then,

$$P(s, a) = p^k \times q^k \tag{5}$$

After that, we also need to calculate the current player position variable D of Mahjong using player position variables and the rule-related transition information:

$$D\left(s^{k-1}, a^k\right) = \left(D\left(s^{k-2}, a^{k-1}\right) + T\left(a^k\right)\right) MOD\ M \tag{6}$$

In Equation (6), $T(a^k)$ denotes the mapping function from the transition information to the change of the player position variable, which depends on the special Mahjong game rules. M denotes the number of players, MOD denotes the modulo operation. Note that we always set the player position variable of the root node to 0, e.g., $D\left(S^0\right) = 0$, and the player position variable is assumed to be in the relative order, such as $0, 1, 2, 3, 0, 1, \ldots$, and so on, which is used to calculate the value estimation for each player.

5. Backpropagation and updating. Finally, we update the statistics of all nodes along the path. For each $k = 1, \ldots,$ leaf, where leaf denotes the depth index of the new leaf node, we compute the change of the player position variable corresponding to the stored variable D:

$$shift^k = \left(D\left(s^{k-1}, a^k\right) - D\left(s^{leaf-1}, a^{leaf}\right) + M\right) MOD\ M \tag{7}$$

Then, we obtain the state-action value of the current edge according to the value vector v^{leaf} and the $shift^k$ value:

$$Val^k = V^{leaf}_{shift^k}, shift^k \in (0, 1, \ldots, M-1) \tag{8}$$

Then, we update the value Q and the edge visits N of the current edge:

$$Q\left(s^{k-1}, a^k\right) = \frac{N\left(s^{k-1}, a^k\right) * Q\left(s^{k-1}, a^k\right) + Val^{k-1}}{1 + N\left(s^{k-1}, a^k\right)} \tag{9a}$$

$$N\left(s^{k-1}, a^k\right) = N\left(s^{k-1}, a^k\right) + 1 \tag{9b}$$

Note that the output range of the value estimation is $[-1, 1]$, for example, the target values -1, 0, and 1 mean loss, no change, and gain of the round score of the game, respectively, which also means the final result of a single round.

3.1.2. Game Information Representation

Since Mahjong has high-dimensional features, we use binary planes to denote each tile and use the column of the plane to denote each type of all tiles. Considering Mahjong contains numbers from 1 to 9 and three suits of characters, dots, and bamboo, we use

27 different tiles and 4 different counts to represent a 27 × 4 two-dimensional matrix realized by a 27 × 1 × 4 three-dimensional plane. This encoding of the tiles' representation allows our agent to better understand the Mahjong features. Figure 2 shows an example.

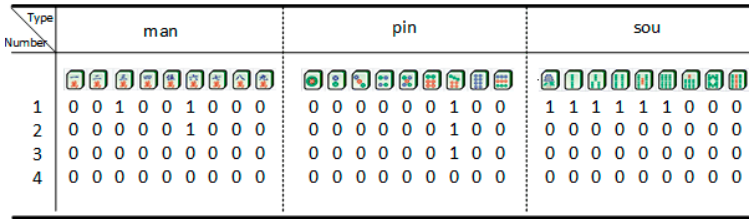


Figure 2. The encoding of Mahjong hand tiles. The horizontal axis represents all possible card types, and the vertical axis represents all possible numbers.

Figure 2 shows a hand of 366 Man (character), 777 Pin (dot), and 123456 Sou (bamboo) tiles.

3.1.3. MDP Model Input

Because Mahjong has complex game rules, we design feature planes with all possible special rules of Mahjong to improve the learning process of the agent. At each step, the player acts according to their hand tiles, other players' public discard tiles, and other features. In addition, we also consider the last tiles and the last action of the latest player. As a result, the following features in Table 1 are used as the input of the representation network. Each feature is encoded by several 27 × 1 × 1 planes for integer features or 27 × 1 × 4 planes for tiles.

Table 1. Input features of representation network.

Features	Number of Channels	Description
Hand tiles of each player	4 × 4	Private tiles of each player
Total discard tiles of each player	4 × 4	Discard tiles of each player
Recent 4 discard tiles of each player	4 × 4	Recent discard of each player
Melds tiles of each player	4 × 4	Melds tiles of each player
Wall tiles	10	Top ten tiles of wall tiles
Current score of each player	11 × 4	Score is divided into 11 integer intervals for each player

Then, we need to design the input features of the dynamic network for Mahjong. In addition to the last internal state, the transition action combined with other information is also provided as an input of the dynamic network. As shown in Table 2, we utilize the tile and action type to specify the basic action information. Since Mahjong is a multiplayer game, the change of the player position variable and the player state are also provided as input. The goal of providing extra information is to improve the accuracy of the value estimation v^k and the prior policy estimation $p^k \times q^k$ of the prediction network. All features are encoded by several 27 × 1 planes and then stacked with the last internal network state.

Table 2. Input features of the dynamic network.

Features	Number of Channels	Description
Tile type	1	Tile type of action
Action type	10	10 types of actions [1]
The change of the player position variable	4	The change of next player's position variable after taking action [2]
Additional player state	4	The next player state after taking action [3]

[1] Actions encoded with one-hot form, as shown in Table 3. [2] e.g., player position +1, +2, +3, +4. [3] e.g., Win, Pong/Kong, Chow, Pass of the player state.

3.1.4. MDP Model Output

There are two types of output estimations: the prior policy estimation $p^k \times q^k$ and the value estimation vector v^k. First, we use a $27 \times 1 \times 55$ convolutional layer to represent all possible actions combined with the transition information. The proposed output design shows faster model convergence and consistent performance compared to a full connection output layer. The detailed output representation is shown in Table 3. Specifically, 48 channels are used to denote the three types of actions that can dynamically trigger one of four possible changes in the player position and one of four possible player states, such as discard, add Kong, and Pass. So, there are $3 \times 4 \times 4 = 48$ different types of dynamic actions. The remaining seven types of deterministic actions have the normal state transition.

Table 3. Prior policy representation.

Action Type	Number of Channels	Action Type	Number of Planes
Discard	4×4	Chow_Right	1
AddKong	4×4	Chow_Middle	1
Pass	4×4	Pong	1
ClosedKong	1	OpenKong	1
Chow_Left	1	Hu (Win)	1

The value estimation vector v^k involves the evaluation value of four Mahjong players, each value within $[-1, 1]$. It also uses the relative order, starting from the player position of the current node. The detailed improvement of the target value for Mahjong is left for future work.

3.1.5. Network Implementation

It can be seen from Figure 3 that the input of the representation network consists of the features mentioned in Table 1 as a three-dimensional $27 \times 1 \times 118$ input layer. The input of the dynamic network consists of the last internal state and transition features mentioned in Table 2 as a stacked input layer, a three-dimensional plane of $27 \times 1 \times (128 + 19)$. The output layers of the prediction network are the two types of estimations mentioned in Section 3.1.4. We do not utilize the internal state directly but just store the network output of the internal state as a representation of a specific Mahjong game situation. The representation and dynamic networks have a main body of 12 repeated residual blocks, with each residual block consisting of two batch normalized, 128 channels, a convolutional layer with a 3×1 kernel, and a prediction network that only serves as computing and outputting two types of estimation. Furthermore, the prediction network outputs P player estimate values, where P is the number of players. In Mahjong scene, P = 4. The stride of all convolutional layers is 1×1. We train all three networks as a joint network, shown in Figure 4, with K = 6 unfolded time steps for each training iteration.

The total training loss of the network is as follows:

$$L_t(\theta) = \sum_{k=0}^{K} \sum_{i=0}^{N} l_i^v \left(Z_{t+k}, v_t^k \right) + l^p \left(\pi_{t+k}, p_t^k \right) + c||\theta||^2 \quad (10)$$

In Equation (10), K = 6 is the unfolded time steps; N = 4 is the number of players; l^v and l^p are the loss functions of the value estimation and prior policy, which are the mean square error (MSE) loss and cross-entropy loss, and $c||\theta||^2$ is the L2 regularization term of the network parameters, which could mitigate the network overfitting problem.

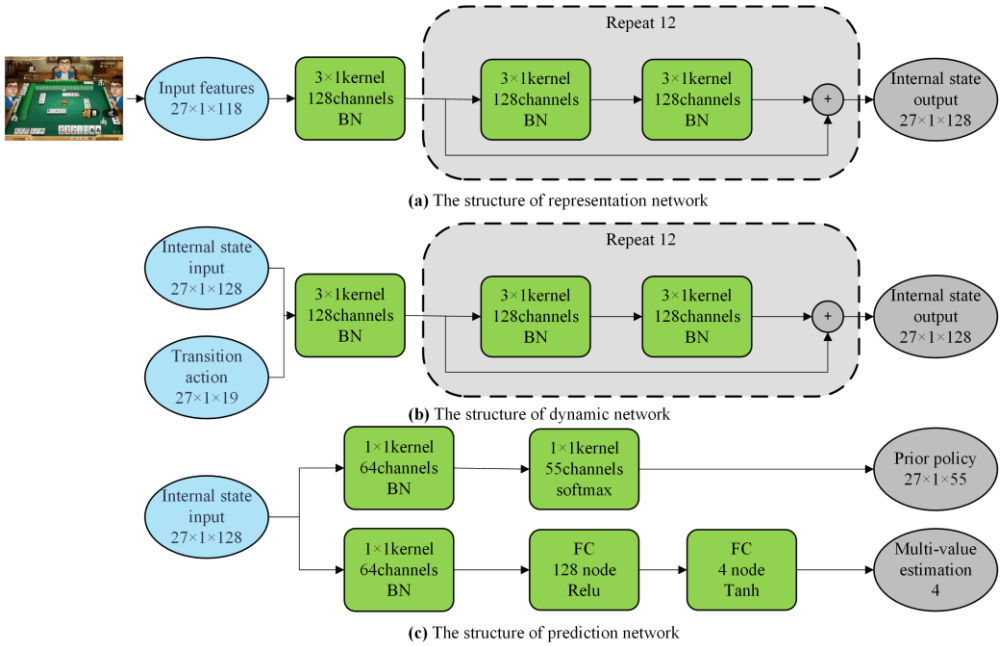


Figure 3. The structure of three networks of the internalized MDP model: (**a**) the representation, (**b**) dynamic, (**c**) prediction network. The representation network uses observed features as input, multiple residual blocks as its main body, and outputs the internal state about the current player. The dynamic networks use a similar structure, but the input layer takes the internal state from the previous step as well as transition actions and information as a combined input. The prediction network is based on different network structures to output corresponding estimated values and assist the tree search process.

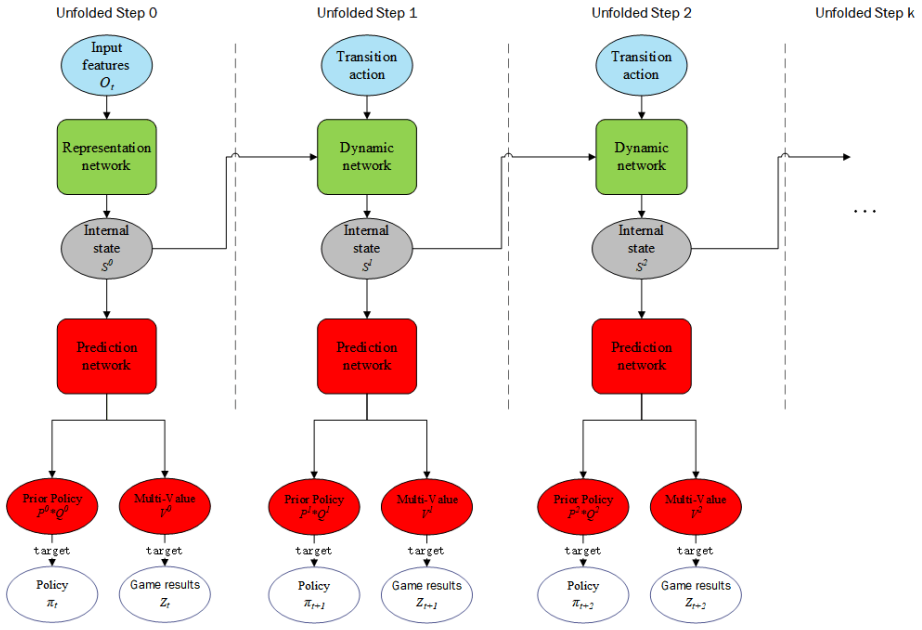


Figure 4. The k-step unfolded training process of the learning network. It uses observation features, transition action, and additional transition information as its input, and then outputs prior policy and multi-value for each step.

3.2. Dropout Learning

After the internalized MDP-based MCTS model is trained completely, then we use the MCTS model to accelerate the training process of the normal agent with imperfect information. Because Mahjong contains a lot of hidden information, if a normal agent

cannot access this hidden information, the performance of the normal agent is more likely to be worse than the MCTS model. Anyway, it is not effective to directly train the normal agent from scratch and we cannot ensure that the normal model can learn the best strategy with a lack of information. To overcome the above problems, we propose dropout learning combined with some improvement approaches.

3.2.1. Learning Algorithm with Bernoulli Random Matrix

Instead of directly training the normal agent from scratch, we use the game strategy data learned and generated by the internalized MDP-based MCTS model as the training dataset of the normal agent. Note that the input of the game strategy data involves all global feature information, but we only want the normal agent to eventually access the normal feature information, such as the player's private tiles, all players' discard tiles, and other public information, during the evaluation. The other players' hand tiles are not available to the normal agent during the normal situation. To achieve this, we combine dropout learning with the special discard features matrix. In detail, we train the normal agent starting from all features with perfect information and then gradually add features with imperfect information. For this purpose, we introduce the Bernoulli random matrix [40] λ and multiply the input layer of the normal agent by the matrix λ:

$$\lambda_{ij} = \begin{cases} 1, & Input_{ij} \in \Phi \\ \sim B(1,p) & otherwise \end{cases} \tag{11}$$

where Φ is a set of normal input features containing all features with imperfect information described in Table 4. $\mathrm{Input_{ij}}$ denotes the input feature in the i-th row and j-th column of the matrix of the input layer. $\mathrm{B(1,p)}$ denotes the Bernoulli function parameterized by p, and λ_{ij} takes a random value depending on this function for each training iteration. Then, we gradually reduce the value p from 1 to 0 with 0.1 for every 60 thousand steps during the training. Finally, $\mathrm{p} = 0$ is fixed during the evaluation. The total training loss is shown in Equation (12):

$$L(\theta) = E_{s,\pi'(s)\sim B}\left[l^p\left(\pi'(s), \pi_\theta(a|\lambda * O(s))\right)\right] + c||\theta||^2 \tag{12}$$

In Equation (12), B denotes the experience replay buffer [41], $\mathrm{O(s)}$ involves all input features described in Table 1 in the state s, π_θ denotes the output policy of the normal agent parameterized by the network θ, $\pi'(\mathrm{s})$ denotes the target policy obtained by the MCTS model in state s, $\mathrm{l^p}$ denotes the cross-entropy loss function, and $\mathrm{c||\theta||^2}$ denotes the L2 regularization term.

Table 4. Imperfect-information features.

No	Feature
1	Player's own hand tiles
2	Discard tiles of each player
3	Melds tiles of each player
4	Current score of each player

3.2.2. Network Implementation of the Normal Agent

The initial input of the normal agent is the representation layer, which consists of the features listed in Table 1; the priority policy of the network output has been simplified from the original 27×55 game actions to 27×10 game actions without additional transition information. The main body of the normal agent consists of 64 residual blocks and is used with 256 hidden channels. The network structure is shown in Figure 5. The agent model consists of an initial feature layer, a main body of residual blocks, and several convolutional layers with batch normalization [42] to output the probability vector of all possible player actions.

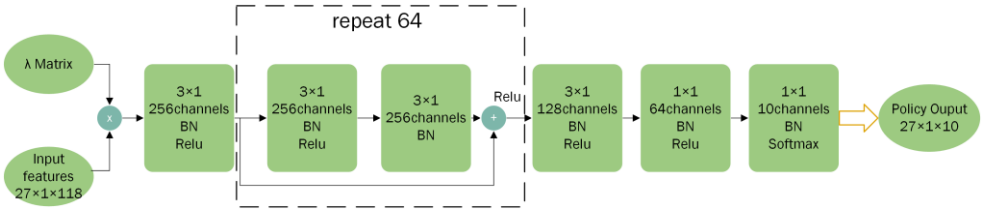


Figure 5. The network structure of the normal agent. It uses the product of the observation features and Bernoulli matrix as its input, then, features are extracted through a 64-layer residual block, followed by additional convolutional layers to reduce the dimensionality and establish a direct connection with the policy output.

3.3. Training Process

The complete training process of the Mahjong agent is divided into two parts: (1) the tree search learning process; (2) the dropout learning process.

First, we train the "perfect agent" with perfect information based on the MCTS-based tree search reinforcement learning. The detailed agent's strategy flow is shown in Appendix B. The brief learning process is shown in Figure A2 and Appendix C, which consists of three parts: the simulation process, the model training process, and the model evaluation process. The detailed tree search Algorithm 1 is as follows:

Algorithm 1 The internalized MDP-based MCTS tree search

Input: O_t: input features vector in step t; θ, μ, σ: network parameters
Output: π: policy probability vector
1. Initialize three functional networks $I_\theta \leftarrow \theta$, $T_\mu \leftarrow \mu$, $E_\sigma \leftarrow \sigma$; set the maximum number of simulations per move K = 250;
2. for $(\text{int i} = 0; \text{i} < \text{K}; \text{i}++)$
3. Node = Root: set the root node as current node; $k = 0$: reset the index of search depth;
4. while Node ! = leaf Node //current node isn't leaf node
5. k++
6. $\text{a}^{\text{k}} = \text{UCB}(\text{Node})$ //determine the next edge according to Formula (2)
7. $\text{Node} = \text{Node}_{\text{a}^{\text{k}}}$ //go to the child node and set it as current node
8. end while
9. if $(\text{Node} == \text{Root})$ //current node is root node
10. $\text{s}^0 = \text{I}_\theta(\text{O}_\text{t}), \text{D}(\text{s}^0) = 0$ //utilize the representation network to compute the internal state of root node and initialize the player position variable of root node
11. else
12. $\text{s}^{\text{k}} = \text{T}_\mu\left(\text{s}^{\text{k}-1}, \text{a}^{\text{k}}\right)$ //utilize the dynamic network to compute the internal state of the child node after state transition.
13. $\text{p}^{\text{k}} \text{x}\ \text{q}^{\text{k}}, \text{v}^{\text{k}} = \text{E}_\sigma\left(\text{s}^{\text{k}}\right)$, store $\text{P}(\text{s}, \text{a})$, $\text{D}\left(\text{s}^{\text{k}-1}, \text{a}^{\text{k}}\right)$, etc. //utilize the prediction network to compute related estimations and initialize related statistics information according to Formula (5) and (6).
14. Backpropagation and update related statistics of each node along the path according to Formula (7)–(9)
15. end for
16. Get the tree search policy π corresponding to the visit count of all edges of the root node.

After the MCTS model is fully trained, we continue to train the normal agent based on the dropout learning approach described in Section 3.2. The detailed training parameters are shown in Table 5.

Table 5. Experiment parameters.

Method	Parameters	Value
Tree Search Learning	Dirichlet_noise_alpha	0.9
	Simulation_count	250
	Total_training_steps	1.2 million
	L2_penalty_alpha	1.0×10^{-4}
	Batch_size	256
	Initial training speed	0.01
	Optimizer	SGD + momentum
	Simulation CPU threads per agent	25
	Simulation GPU threads per agent	5
	Data generation agent used	10
Dropout Learning	Batch_size	512
	Channels	256
	Initial_learning_rate	0.001
	L2 penalty item	1.0×10^{-4}
	Momentum_alpha_beta	0.9, 0.99
	Total_training_steps	0.66 million
	Dropout_decay_rate	0.1
	Dropout_decay_steps	60 thousand

In our actual experiments, the Mahjong simulator code runs in Python to facilitate cross-platform operation, while the agent model and the tree search code are implemented by C++ programs to improve decision-making efficiency and implement concurrence on multiple threads. The interaction between the two is realized through Python's embedding technology.

4. Experimental Results and Analysis

In this section, we validate the learning process of the internalized MDP-based MCTS model, the dropout learning with and without the Bernoulli random matrix. In addition, we test the normal agent on the online game platform of Mahjong with human players and conduct tournaments with other AI models to demonstrate the performance of the proposed agent.

4.1. Experiment Settings

The experiment environment of this paper is IntelCore (TM) i7 12700 CPU, 32 GB memory, Nvidia Tesla V100, and Windows 10 64-bit operating system. The experiment code is written in C/C++ and Python, and implemented based on Cuda, Cublas, and other libraries.

The Mahjong test data is obtained from the Mahjong platform of JJ World Game Company. In addition to removing the faulty game records and the defective data, we also use the data of the game records where the score of the players is more than 5000 points. The score represents the player's performance of playing Mahjong. The more points, the better the player's level. After removing the data of players with poor performance, there are about 400,000 state-action pairs of Mahjong played by human players. According to the total number of game data pairs, we divide them into a verification set and a test set in a ratio of 1:2. To reduce the influence of other factors, we conduct 10 independent experiments and use the average value of them as the final result; each independent experiment is performed with a different random number generator. For the MCTS-based model, the corresponding action with the highest number of edge visits is selected as the output action; for the normal agent model, the action with the highest probability is selected (if there are multiple available actions with the same probability, then one of them is randomly selected). The detailed experiment configurations are shown in Appendix D.

4.2. Model Performance Validation

First, we test the MCTS-based models with different simulations during the RL training. All models use the same internalized MDP network as described in Figure 3, and the same training parameters mentioned in Table 5, except for different simulation counts: 10, 50, 100, and 250 per move during training. Furthermore, all models utilize the same 500 simulations per move during the evaluation of the Mahjong test data. The training loss of the models with different simulations is shown in Figure 6a. The accuracy rate on the Mahjong test date set of models with different simulations is shown in Figure 6b.

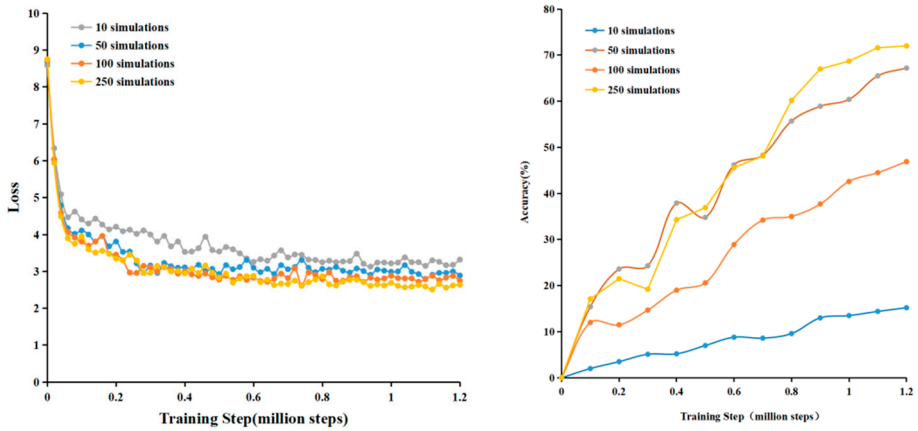


Figure 6. Models with different simulations. (**a**) Training loss of different models over training steps. (**b**) The accuracy rates of different models over training steps.

Figure 6a shows that the training loss starts to decrease for all models and the accuracy rate of all models slowly increases from 0.2 million steps. The accuracy rate on the test data is improved during the RL training, however, the growth rate continues to decline. At the beginning of training, the loss of the 250-simulation per move model is slightly lower than that of the other models. As training progresses, the training loss of the 250-simulation model is obviously lower and smoother than that of all the other models, this is because the 250-simulation model not only learns the prior policy estimation well but also predicts more accurately the value estimation. The experiment results show that the 250-simulation model finally has the lowest training loss and the 100-simulation model is second. Further, it can be seen from Figure 6b that models with 50 or fewer simulations cannot learn game strategy very well, and their accuracy rates are lower than 30% after half of the training steps. The model with 10 or 50 simulations per move could not reach a 50% accuracy rate after the complete training. In contrast, when given more than 50 simulations per move during training, the accuracy rate of the model has a great improvement. The more simulations, the higher the magnitude of accuracy increase. The highest accuracy rate reached is 72%, with 250 simulations per move. The results show the effectiveness and stability of the internalized MDP-based MCTS model on Mahjong test data.

To better describe the effects of feature access on the performance of the MCTS-based model, we compare the MCTS-based models with access limitations on different features. The original version could access all feature information, and we also designed two variants based on the original model: one of which removes all unconventional input features, such as other players' private hand tiles, and the other replaces all unconventional input features with random features instead of empty features. Both models utilize 250 simulations per move during the training. The detailed experiment setup for both models are mentioned in Section 3.3. The results comparing the accuracy rate between the three different models over different evaluation simulations per move are shown in Figure 7a.

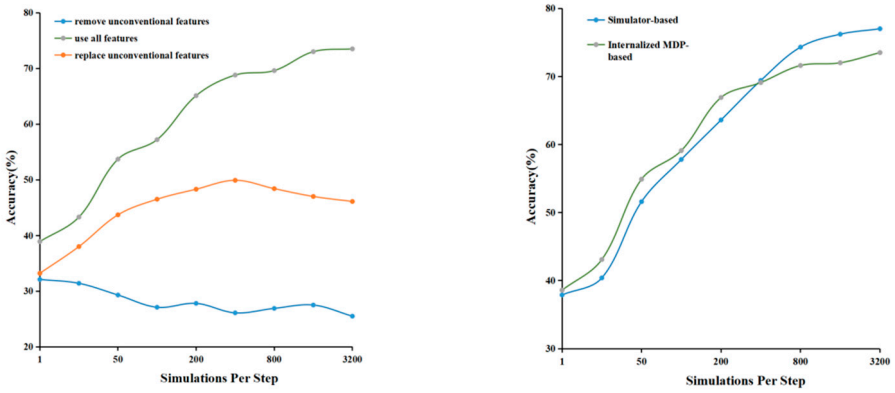


Figure 7. Effect of different factors on model performance. (**a**) Effect of feature access on model performance over simulations. (**b**) Effect of internalized MDP network on model performance over simulations.

We can see from Figure 7a that the accuracy rate of the model with all of the unconventional input features removed could not reach 35%, and cannot be improved by increasing the evaluation simulations. In contrast, the other model, replacing unconventional features with random features, has a better accuracy rate, although it is still lower than that of the all-features-access model. This is because the former could utilize additional random features instead of empty features to improve the tree search process. Regardless of whether the random features information is correct or not, the game strategy can be improved to a certain extent through the value-averaging approach of tree search. Compared with the other two versions, the accuracy rate of the all-features-access model still increases as the number of simulations increases. Finally, it reaches 72% and is better than the other two variants. This comparison confirms our inference that the feature access has a great influence on the performance of the MCTS-based model. Further, to verify whether the internalized MDP network improves on the performance of the MCTS-based model, we train and compare two different versions: one is based on the internalized MDP network and the other is based on the accurate Mahjong simulator (it can be equivalent to utilize the representation and prediction network to calculate the prior policy and value estimation of nodes, and it does not implement the internal state transition of Mahjong) over different evaluation simulations. The accuracy rate of the two models is shown in Figure 7b. When less than 500 simulations of evaluation are used, the accuracy rate of the internalized MDP-based model is higher than that of the accurate simulator-based model, except when using a single simulation, which is the same as directly using the prior policy as model output. However, when the number of simulations is more than 500, the magnitude of accuracy increase in the internalized MDP-based model is suddenly decreased but the accuracy rate of the simulator-based model can still steadily increase. By analyzing the MDP model, the possible reason is that, as the number of simulations increases, the depth of the nodes will also grow and the internalized MDP-based model is more likely to compute a noisy value estimation, due to the limitation of the recurrent neural network structure and the precision of the matrix multiplication operations, etc. In short, the experiment results indicate that the internalized MDP model could still make effective improvements on the model performance under a certain number of simulations.

The test accuracy of the models is reported in Table 6. The MCTS-based model utilizes 134 thousand validation data and 266 thousand test data. Considering the time and computing cost, the simulator-based and MDP-based MCTS models both use 250 simulations per move during evaluation. In addition to the above models, we also report the accuracy rate obtained by other models in previous works [43] for comparison. It can be seen from Table 6, that we obtain a 69.6% average accuracy rate for the simulator-based MCTS model and a 72.0% average accuracy rate for the MDP-based MCTS model. However, we do not

achieve the highest accuracy among all models and more research could focus on improving the target game reward to obtain higher data accuracy. Note that due to different test data and different Mahjong game rules, we just list the data and indirectly compare our models with previous works.

Table 6. Accuracy results.

Model	Highest Accuracy	Mean Accuracy
Simulator-based MCTS Model	70.5%	69.6%
MDP-based MCTS Model	73.0%	72.0%
Previous Works	88.2%	68.8%

4.3. Offline Experiment

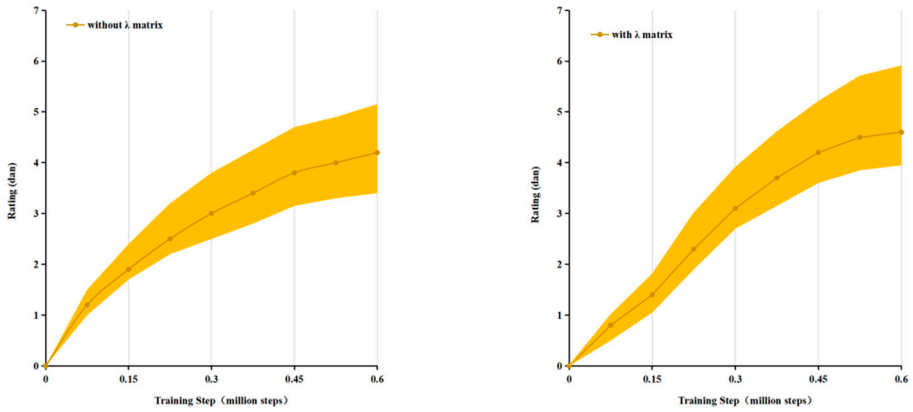
In this section, we conduct offline tournaments between the learning-based normal agent and the open-source Mahjong engine. We use the public open-source Mahjong engine as a baseline player for offline evaluation. This Mahjong engine, called Tenhou-Bot (https://github.com/MahjongRepository/tenhou-python-bot (accessed on 12 July 2022)), which is programmed with a hand-designed value function, has conducted a large number of anonymous online confrontation tests on the online Mahjong website Tenhou.net, with a total of about 1000 online games and the final rating is between three dan and four dan (the ranking system adopted by Tenhou.net, https://tenhou.net/man/ (accessed on 15 July 2022), which starts from rookie, 9 kyuu down to 1 kyuu, and then 1 dan up to 10 dan), with a first place rating of 23.65%. To be compatible with the Mahjong rules in our paper, we remove the extra seven tiles (four winds and red, green, and white tiles) and make some additional modifications to the winning condition on the above engine. To ensure the fairness and effectiveness of the evaluation, the agent ranking adopts the standard ranking formula proposed by Tenhou:

$$Rank = \frac{5 * n_1 + 2 * n_{-2}}{n_4} - 2 \tag{13}$$

In Formula (13), n_1, n_2, and n_4 represent the total times of the first, second, and fourth rankings, respectively. The ranking rating is computed from the results of the tournament between iterations of the dropout-learning-based agent during training and the baseline player (engine). Since Mahjong has a lot of random feature information, such as the initial player hand tiles, it is generally assumed that not less than a hundred games are needed to obtain a stable rank. So, we set up 10 random independent experiments, each has 100 random games, each game has four rounds, and we use the average ranking rating as the final result. We randomly select three of the four players as the baseline player in each game.

First, we compare the effects on the performance of the agent with and without the discard feature matrix during training, with the same training data obtained by the internalized MDP-based MCTS model. All parameter settings and training processes of both models are the same, except one is combined with the Bernoulli random matrix, the other does not directly use it and only takes conventional features as input by setting $p = 0$ throughout training. The number of training steps for both models is set to about 0.6 million steps. The training results are shown in Figure 8. The ranking results are shown on the y-axis, which is calculated corresponding to Formula (13), millions of training steps on the x-axis. The dark line denotes the median score across 10 separate experiments and the shaded region denotes the 25th to 75th percentile. We can see from Figure 8 that at the beginning, the rating perform of the agent with the Bernoulli random matrix is lower than that of the agent without it because the former needs more time and steps to extract the related features from randomly dropping features during the training iterations. As training continues, the rating of the agent with the Bernoulli random matrix begins to be higher than that of the agent without it. This is because it is hard for the agent without

feature dropping to establish the network connection between the conventional features and the target action learned by the "perfect model" without the help of additional feature information, such as the hand tiles of other players. From another perspective, it can be explained that the model has not been pre-trained for enough time to better initialize the network parameters, making the network easier to overfit and thus converge earlier, and it finally falls into the local strategy. As a result, it cannot be trained very well and only reaches an average rating of about 4 dan. In contrast, the agent with the feature dropping achieves an average rating of about 5 dan and a highest rating of about 6 dan. However, its rating cannot be further improved even given more training steps. There are two possible reasons: one is that the training data itself is not perfect, and the learning algorithm of the MCTS model needs to be further improved first; the other is that the hyperparameters in the training affects the convergence performance of the agent, and future investigation could focus on adjusting the learning rate and the decrease rate of the parameter p to achieve a better rating performance of the normal agent.



(**a**) (**b**)

Figure 8. Effect of the discard feature matrix on the rating of the agents during training. (**a**) Agent without Bernoulli random matrix. (**b**) Agent with Bernoulli random matrix.

Finally, we report the ratings of our agent and baseline player in Table 7, and we also list the Mahjong model proposed by other researchers [28] as a reference. We can see that our model is 2 dan higher than the baseline player Tenhou-Bot and very close to the Bakuuchi AI engine, with fewer games, and also the professional human player, which indicates a certain effectiveness of our proposed agent.

Table 7. Ranking comparison with different Mahjong engines or human player.

AI Model/Engine	Number of Games	Stable Rank
Proposed agent	3328	5 dan
Baseline player Tenhou-Bot	1095	3 dan
Bakuuchi	30,516	6 dan
Professional human player	-	5 dan

4.4. Online Experiment

In this section, we conducted the Mahjong test on an online Mahjong platform. The purpose is to make a good-performance Mahjong agent that can replace the human player to take action and win the game. One player is the proposed agent during the online evaluation, and the other three are human players.

It can be seen from Figure 9 that the agent draws six character tiles and makes a decision to discard one of the hand tiles. Due to the winning tiles combinations of Mahjong, and the most similar type of player hand tiles is normal, the agent is ready to discard the unnecessary single six characters. The program of our agent is shown on the left-hand side

of Figure 9. As the input features are passed to the program, the agent outputs the fifth action (starting from 0), which exactly means the discard of six characters. We can see that our agent takes the discard action effectively and follows the rules of the Mahjong game.

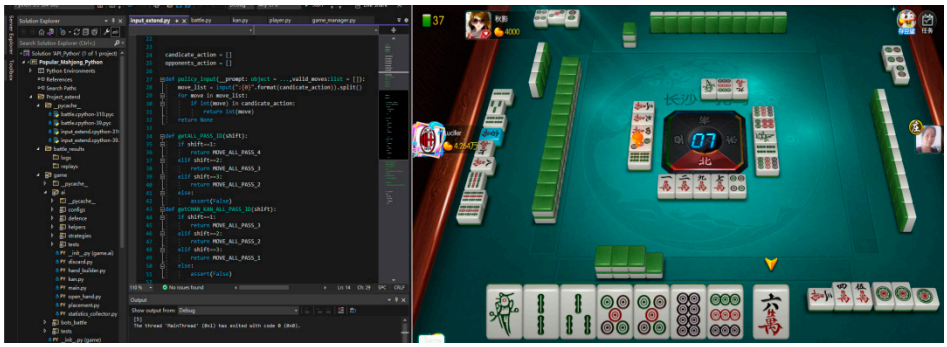


Figure 9. Test example: discard tiles.

Figure 10 shows a certain situation of Mahjong. In the above situation, the available action Hu will end the game immediately and obtain the score points from other players, and the Chow action will make a meld and continue the game. In most situations, the player should choose Hu if possible. So, our agent decides to take Hu action and outputs the 258th action (the Hu action type with 7 dots) according to Figure 2 and Table 3, which corresponds to the Hu of 7 dots. This shows that our agent knows the winning conditions of Mahjong and takes the Hu action if possible.

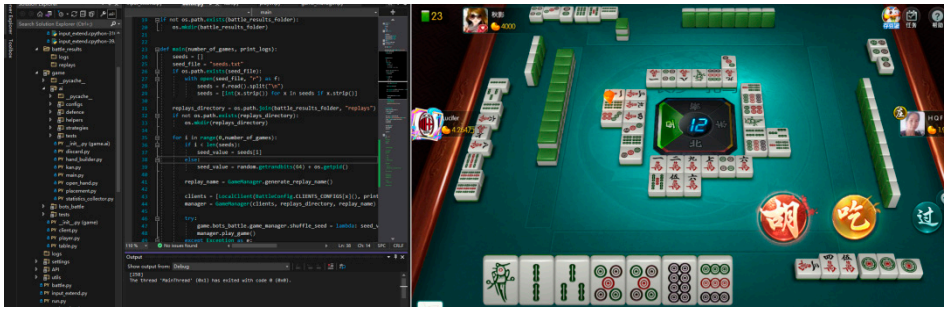


Figure 10. Test example: declare winning hand.

As shown in Figure 11, after the left player discards 8 dots, it is time for the agent to choose whether to take Chow or Pong or Pass. Because it has three different types of tiles and Pong 8 dots, which do not influence the other two types of tiles, our agent decides to make a Pong meld and speeds up the progress to a winning hand. In addition, we can see that the program of the agent receives the information of the game situation and is ready to take the 205th action after seconds of computing, which is the Chow of the 8 dots according to Table 3. It shows that the agent will make a meld in the proper situation and act like a human player.

In the situation of Figure 12, the last player discards 7 dots, and our agent has two 7 dots. If we choose Pong, it will not disassemble the hand tiles. However, the agent outputs the 69th action and decides to choose Pass instead of Pong (204th action), which will miss an opportunity to make a meld quickly. In view of lacking information, the possible reason is that there are too few discarded tiles and our agent hesitates to take Pong, which shows that the agent may not have a good understanding of when to take Pong.

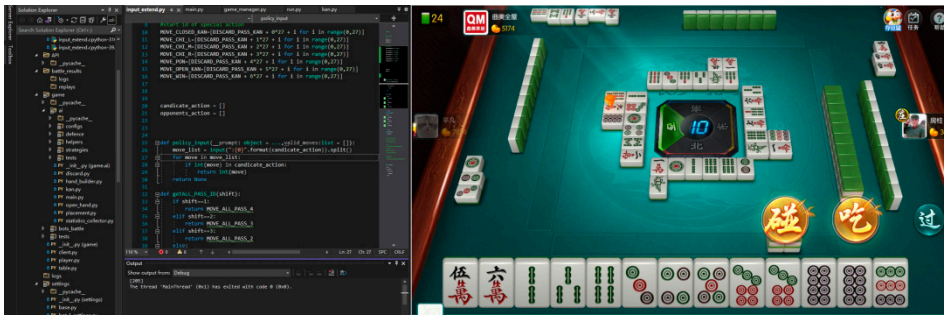


Figure 11. Test example: make a meld.

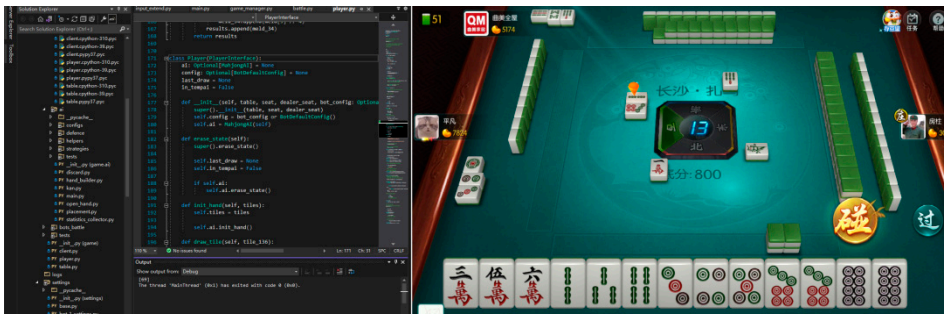


Figure 12. Test example: the other situation.

To learn about the more detailed winning strategies, the amounts of different winning scores obtained by our agent are shown in Table 8, where one, two, four, and eight mean the corresponding scores multiplier. The occurrence of the one multiplier is very frequent, because the agent does not consider the score factor as one of the learning objectives, so the strategy is more inclined to the strategy that can win quickly, and usually the strategy with very few points. The more rounds of the game, the greater the impact. Therefore, in the future work, the factor of score should also be taken as one of the goals of the model learning.

Table 8. The distribution results of the winning scores.

Score Multiplier	One	Two	Four	Eight	More
Total	378	38	8	3	1
Occurrence Rate	88.3%	8.9%	1.9%	0.7%	0.2%

In addition, we also participated in the Mahjong group of the 2022 Competitive World Cup national computer game tournament [44] and eventually won third place after combining the proposed agent with other improvement methods. A total of 16 teams were divided into four groups for a total of three rounds. The first-place and second-place teams in each group advanced to the next round of the final round, which was scored based on total points. Our Mahjong agent successfully finished second in the group and moved on to the final round, where it eventually finished third, indicating some performance and efficiency.

5. Discussions and Conclusions

In this paper, we present a new AI learning method and successfully train a Mahjong AI agent to demonstrate the convergence and effectiveness of our method. Different from the existing learning methods, this work first takes advantage of the internalized MDP model and multiplayer MCTS algorithm to obtain the improved game strategy from model-based reinforcement learning. With the almost "perfect" game strategy and the Bernoulli random matrix, the proposed dropout-learning-based normal agent enables faster

convergence, better performance, and increased effectiveness in the imperfect-information Mahjong game. The experiments demonstrate the good performance of the proposed MCTS-based model by comparing between different simulation models during RL training. To further verify the effect of feature access and the internalized MDP model on the performance of the model-based learning approach, the models are analyzed with different influencing factors over evaluation simulations. Moreover, the proposed agent is evaluated against the other baseline engine. The results show that the agent trained by our methods achieves a good ranking performance and can play like a human in the Mahjong game.

However, the proposed methods and agent also have some limitations. First, it is very computationally intensive due to the huge search tree and the large state-action space of Mahjong. Second, it is currently only validated for Mahjong and it is not certain whether the proposed methods can be applied in any other game scenarios. Future work will focus on expanding the proposed methods to more complex and general game scenarios and scaling it for training agents under multiple scenarios with optimization of the reward signal. This will be more challenging, as the dimensions of the state and action space will increase exponentially.

Furthermore, the proposed agent utilizes the round result of the game as a learning signal, ignoring the detailed winning score information. The utilized learning signal is too simple, and it is difficult to distinguish the different performances of the different winning scores. Utilizing the additional reward predictor, such as the global reward prediction model, has the potential to reduce the variance of the learning signal for each round and optimize the learning signal by combining the winning score information and the final game result, and further improve our agent.

When faced with more complex game scenarios, how to better represent the current state, transition actions, and the impact of the state transition process in the network is particularly important. In addition, how to better integrate it with convolutional networks or combining the characteristics of other learning networks is also the key to further improving the performance of the model. The feature representation of other games will obviously affect the final model performance. In the dropout learning, in the face of different games, the more hidden information there is, the more difficult it is to train the agent model and the more training time is required.

The proposed method mainly combines several general agent training approaches and the learned network still requires some necessary prior knowledge related to the specific environment. For other multiplayer games, if more prior knowledge and transition information related to the performance of the agent are provided during the state transition, a corresponding better performance can be obtained in theory, and the agent learning method has a certain generalization. Overall, the proposed method gradually constructs the agent model by decomposing the decision-making problems related to specific scenarios, which has certain adaptability and can also be applied in related fields such as robot control and human–computer interaction, providing a new AI agent-based modeling idea.

Author Contributions: Conceptualization, T.Z. and X.O.; methodology, T.Z.; software, T.Z.; validation, X.O.; formal analysis, X.O.; investigation, T.Z.; resources, X.O.; data curation, T.Z.; writing—original draft preparation, T.Z.; writing—review and editing, X.O.; visualization, T.Z.; supervision, X.O.; project administration, X.O.; funding acquisition, X.O. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data is unavailable due to privacy.

Conflicts of Interest: The authors declare no conflict of interest.

Appendix A. Mahjong Game Rules

Mahjong is an attractive multiplayer game in China. It has the same basic rules as most types of Mahjong, such as Bloody Mahjong. However, it still has some special game rules. The detailed rules are shown below.

1. Terminology

Definition 1—Ready Hand

This refers to the game state when the player can win only with another tile. According to the game rules, the player can choose whether to declare or not, after declaring ready hand, the player's game will obtain points rewards and enter the auto mode, that is, except for the behavior of calling Win and Kong, they can no longer perform any other actions, such as changing hand tiles, that is, they must discard whatever tiles they draw, Chow, Pong, etc.

Definition 2—Chow

Chow tiles mean that the player has two tiles adjacent or one apart. When the previous player discards the adjacent or middle tiles, they can chow that card and make a meld.

Definition 3—Pong

Pong tiles mean that there is a pair of identical tiles, when any other player discards the same tile, the player can Pong this tile.

Definition 4—Open Kong

A type of Kong which means that the player has three identical tiles. When any other player discards the fourth tile of this type, the player can Open Kong this tile, which is also called the straight Kong.

Definition 5—Closed Kong

A type of Kong. When a player has four identical tiles in their hand, they can Close Kong with these four tiles. The difference from the straight Kong is that the fourth tile of the Kong is obtained by the player themselves from the wall.

Definition 6—Add Kong

After the player makes a meld, they then draw the fourth tile that is the same as the meld tile, and they Kong this tile, that is, an Add Kong.

Definition 7—Hu (Win)

When the 14 cards in the player's hand can form a tile type with specific combination conditions, it is called Hu (Win). This combination condition is the biggest difference between Mahjong in different regions and different ways of playing.

2. Game rules

In the beginning, each player starts with 13 tiles, but the dealer plays the first tile, so they hold 14 tiles. Then, players discard tiles counterclockwise. To win the game, players need to constantly combine, split, and recombine, depending on the tiles drawn. The goal is to quickly form the 14 tiles in their hands into a specific combination. Based on the Formula (A1a,b), players also try to reach the maximum score indicated in Table A1.

Table A1. Winning hand calculation.

Type	Multiplier	Number of Suits	Tiles Combination
X-normal (Ping Hu)	6	3	$(4-a) \times AAA + a \times ABC + DD$
Y-bump series	8	3	$a \times AAA + DD$
Z-pure series	12	1	$(4-a) \times AAA + a \times ABC + DD$
Q-seven pairs (Qi Dui)	12	3	$7 \times DD$

$$(4 - a) \times AAA + a \times ABC + DD, \quad 0 \leq a \leq 4 \tag{A1a}$$

$$b \times DD, \; b = 7 \tag{A1b}$$

Appendix B. Strategy Flow of Agent Model

As shown in Figure A1, we start by building the model's strategy flow. It has two main situations which it requires the agent model to handle. In particular, in the main step, we integrate all actions instead of making multiple strategy models. We use the action mask provided by the Mahjong simulator only in the root node to remove invalid actions and improve the efficiency of action selection. Further, in the other-discard step, since the interrupting actions, e.g., Pong, Chow, have action priority, the priority will be made according to the priority order of Ron (win) > Kong or Pong > Chow. The action priority is also provided by the Mahjong simulator and the agent model gradually learns the action priority through training.

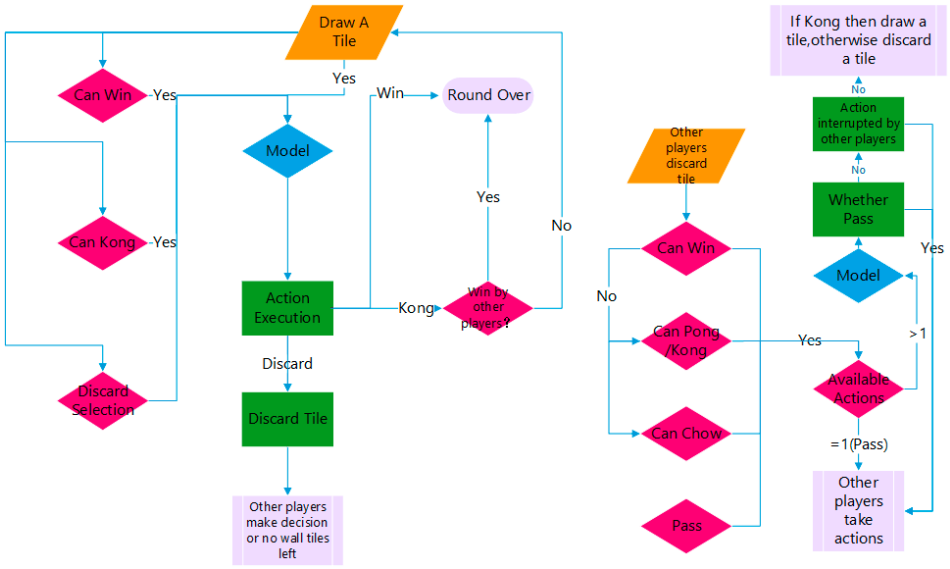


Figure A1. Schematic diagram of model main step and other-discard step.

Appendix C. Implementation of Tree Search Algorithm

The RL model learning process is shown in Figure A2 and consists of two parts: interaction with the simulator, model training, and evaluation. The details are as follows:

- Simulation: Playing data are generated through the internalized MDP-based MCTS model that interacts with the Mahjong simulator. Specifically, 250 simulations in each step during training and the first 15 actions are randomly sampled from the policy in order to extend the coverage of trajectory data. For the remaining steps, the action with the highest probability is selected. Furthermore, we also add Dirichlet noise Dir(α) [45] to the prior policy of the root node to encourage the exploration of new strategies, which is related to the expected number of actions or moves. So, we have:

$$P(s, a) = 0.7 * P(s, a) + 0.3 * Noise(s, a) \tag{A2a}$$

$$\boldsymbol{Noise}(\boldsymbol{s},\boldsymbol{a}) \sim \boldsymbol{gamma\_dist}(\mathbf{10}/\boldsymbol{legal\_moves\_count},\mathbf{1}) \quad \text{(A2b)}$$

In the above equation, $\alpha = 10/legal\_moves\_count \approx 0.9$ in our experiment in order to randomly sample about 10 moves over all legal moves. In addition, we also randomly sample actions of the other-discard step in order to further improve the Mahjong game trajectory data.

- Training: Networks were trained with stochastic gradient descent and momentum item [46], and the annealing method is also used to gradually reduce the learning speed, from an initial speed 0.01 to 0.001 and 0.0001 after the 10th and 20th iterations of the model version, respectively. The batch size is 256, momentum item parameters: alpha = 0.9, beta = 0.99. All of the training data follows the experience replay buffer design, which always stores the training data of the latest several model versions and regularly removes outdated training data.
- Update: For each model version update, the current version is matched against the last best version with a thousand games. Then, we compare the winning rate of the two models to obtain the best version among them.

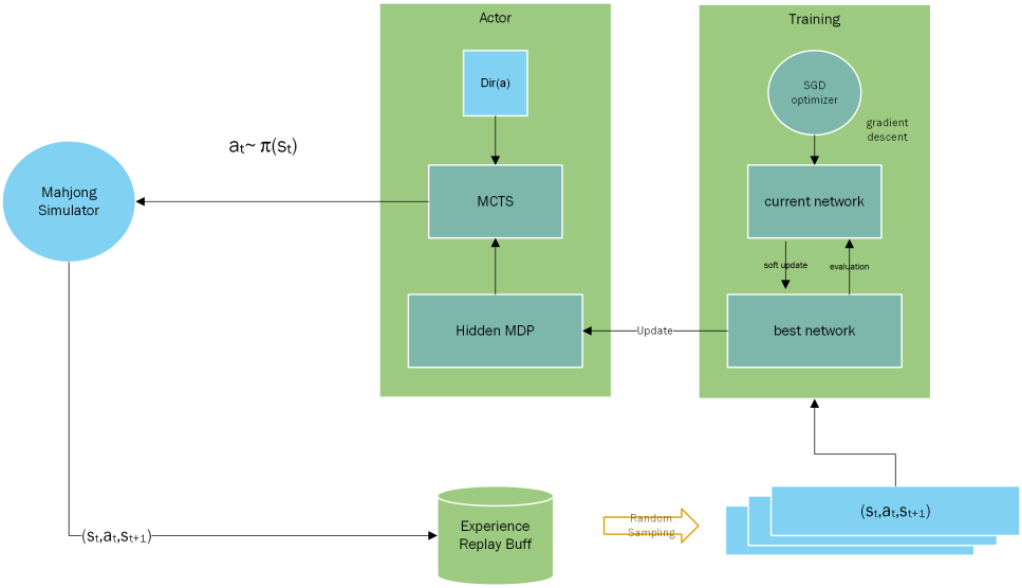


Figure A2. Tree search process of reinforcement learning. This mainly consists of a simulation process and a network training process.

For each model version, we performed steps 1, 2, and 3 in sequence to obtain the current best network, and then updated the MDP network of MCTS with the best network. After that, we repeated steps 1, 2, and 3, and so on.

Furthermore, in order to speed up model training, virtual loss [47] technology was also used in step 1 to improve the performance of multi-thread asynchronous simulations within the search tree. Similarly, multi-CPU tree search and multi-GPU network prediction concurrency technology were used to asynchronously execute multiple simulations to increase the efficiency of the data generation. In step 2, the models were trained with the DistBelief [48] technology to asynchronously update the gradient to speed up the network convergence.

Appendix D. Experiment Configurations

Test Data Collection: The Mahjong test data is obtained from the Mahjong platform of JJ World Game Company. We divide the collected test data into a verification set and a test set in a ratio of 1:2. The original data are game logs generated by human players. These logs consist of all information of each player and the wall tiles. We first need to parse the log strings to standard ascii arrays, then translate them to the corresponding tiles and actions information. For example, 0(0)8348346:0203040912131622232425282828 is the original log which contains a player's basic information, the first number, 8348346, is the current player's points and the second number string is the initial hand tiles of this player. The obtained ascii arrays, such as 3D01, means the 3rd player discarded the one character: D represents discard, P represents Pong, and H represents win, and so on. After parsing these log data, we also need to remove data with errors or incomplete records. An example log of a Mahjong game is shown in Figure A3.

```
3(1)6938143:030608111212151617182122232 6;
0(0)8348346:020304091213162223242528 28;
1(1)5998331:020508121314151622232425 26;
2(1)2192733:010405051113171920202121 27;
3D060M290D151P151D162D163M053D050M280D293P293D280M120D281M111B111M231D232M182D183M073D070M260D121M141D143P143D260M190D19
```

Figure A3. Example log data of a game round.

Tournament Configuration: To evaluate performance, we used the open-source Mahjong AI engine (https://github.com/MahjongRepository/tenhou-python-bot (accessed on 12 July 2022)) as a baseline engine. Tenhou-Bot was a good performance (3~4 dan) hand-designed AI based on python and third-party Mahjong API. The author carefully designed a value estimation system based on score estimation, and made strategic priority adjustments for all possible situations. We made some adaptive modifications on the basis of it, such as being compatible with the agent model (C++) through Python embedding technology and capable of interacting, making detailed adjustments to the program code to adapt to the simulated Mahjong environment of this article, and repairing bugs in some special scenarios. The baseline engine only required a single CPU to run, and each step in the decision-making process could be controlled within a completely acceptable running time (<0.5 s/step). Then, we measured the real performance of the proposed agent model through game tournaments against the above Mahjong engine. We set up 10 independent tournaments for evaluation to reduce the effect of the randomness of initial hand tiles, each tournament consisted of 100 Mahjong games, each Mahjong game consisted of four rounds. In each game, we randomly selected one of four players as the agent model and the other three of the four players as the above Mahjong engine. The Tenhou stable rank rule, which was mentioned in Section 4.3, was used to fairly evaluate the performance of the proposed agent model. The agent model took about 0.5 s per step (move) under our hardware conditions. The opponent took about 0.3 s average per move and did not use any search or network evaluation.

Online Settings: Based on the online Mahjong game platform, the development of the agent is realized according to the API interface document in the Python 3.9 environment. As an object-oriented interpreted programming language, Python is highly scalable and has a rich and powerful class library. In addition, it also has the advantage of being cross-platform.

Tiles representation: Suit: consists of Pin (P), Sou (S), Man (M), other tiles if included (Z). Rank: 1, 2, 3, 4, 5, 6, 7, 8, 9. Among them, suit and rank can be combined with each other, for example, 4S means four Sous, and 3M means three Mans.

Send message: The server interacts with the client and sends messages. The server will always monitor the actions of all players in the game, collect and update the status information during the game, encapsulate the information in json data format, and then send it to all clients in a timely manner. The API interface definition is shown in Table A2 below.

Table A2. API interface definition.

Key	Type	Description
Pong	string	Pong meld, e.g., "222M"
Chi	string	Chow meld, e.g., "234S"
Kong	string	Kong meld, e.g., "3333S"
Seat	string	Current player seat, e.g., "1"
History	array	Action or move history sequence, e.g., "["1,Chi,234S","1,Discard,5M"]"
Hand	string	Player hand tiles, e.g., "33M2468P6678S123Z"

Response: client feedback action command. After receiving the json data information sent by the server, the client first uses the seat field to determine whether they are the current decision-making player. If they are, the client uses the relevant data of the server for calculation and inference, and then uses the API interface to return the data in the specified format; otherwise, the client will ignore this information. The definitions of data sent by the client are shown in Table A3 below.

Table A3. Client response definition.

Key	Type	Description
Code	int	http status code, e.g., 100
Err_msg	string	error message, e.g., "illegal"
Action_content	string	tiles of action, e.g., "123M"
Action_type	string	type, e.g., "Pong"

References

1. El Rhalibi, A.; Wong, K.W.; Price, M. Artificial intelligence for computer games. *Int. J. Comput. Games Technol.* **2009**, *2009*, 251652. [CrossRef]
2. Bourg, D.M.; Seeman, G. *AI for Game Developers*; O'Reilly Media, Inc.: Sebastopol, CA, USA, 2004.
3. 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. [CrossRef] [PubMed]
4. 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. [CrossRef] [PubMed]
5. He, K.; Zhang, X.; Ren, S.; Sun, J. Deep residual learning for image recognition. In Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA, 27–30 June 2016. [CrossRef]
6. Zheng, J. *Research and Application of Computer Games with Imperfect Information*; South China University of Technology: Guangzhou, China, 2017.
7. Brown, N.; Gross, A.L.S.; Sandholm, T. Deep counterfactual regret minimization. In Proceedings of the 36th International Conference on Machine Learning, ICML 2019, Long Beach, CA, USA, 10–15 June 2019.
8. DeepMind. AlphaStar: Mastering the Real-Time Strategy Game StarCraft II. DeepMind. 2019. Available online: https://deepmind.com/blog/article/alphastar-mastering-real-time-strategy-game-starcraft-ii (accessed on 1 June 2022).
9. Statt, N. OpenAI's Dota 2 AI Steamrolls World Champion E-Sports Team with Back-to-Back Victories. The Verge. 2019. Available online: https://www.theverge.com/2019/4/13/18309459/openai-five-dota-2-finals-ai-bot-competition-og-e-sports-the-international-champion (accessed on 10 June 2022).
10. Shahbazi, Z.; Byun, Y.-C. Agent-Based Recommendation in E-Learning Environment Using Knowledge Discovery and Machine Learning Approaches. *Mathematics* **2022**, *10*, 1192. [CrossRef]
11. Platas-López, A.; Guerra-Hernández, A.; Quiroz-Castellanos, M.; Cruz-Ramírez, N. Agent-Based Models Assisted by Supervised Learning: A Proposal for Model Specification. *Electronics* **2023**, *12*, 495. [CrossRef]
12. Heidari, A.; Jamali, M.A.J.; Navimipour, N.J.; Akbarpour, S. A QoS-Aware Technique for Computation Offloading in IoT-Edge Platforms Using a Convolutional Neural Network and Markov Decision Process. *IT Prof.* **2023**, *25*, 24–39. [CrossRef]
13. Amiri, Z.; Heidari, A.; Navimipour, N.J.; Unal, M. Resilient and dependability management in distributed environments: A systematic and comprehensive literature review. *Clust. Comput.* **2023**, *26*, 1565–1600. [CrossRef]
14. Dao, G.; Lee, M. Relevant Experiences in Replay Buffer. In Proceedings of the 2019 IEEE Symposium Series on Computational Intelligence, SSCI 2019, Xiamen, China, 6–9 December 2019. [CrossRef]
15. Cortes, C.; Mohri, M.; Rostamizadeh, A. L2 regularization for learning kernels. In Proceedings of the 25th Conference on Uncertainty in Artificial Intelligence, UAI 2009, Montreal, QC, Canada, 18–21 June 2009.
16. Hoffer, E.; Banner, R.; Golan, I.; Soudry, D. Norm matters: Efficient and accurate normalization schemes in deep networks. *Adv. Neural Inf. Process. Syst.* **2018**, 31. [CrossRef]
17. Andrew, A.M. Reinforcement Learning: An Introduction. *Kybernetes* **1998**, *27*, 1093–1096. [CrossRef]

18. Sutton, R.S.; McAllester, D.; Singh, S.; Mansour, Y. Policy gradient methods for reinforcement learning with function approximation. *Adv. Neural Inf. Process. Syst.* **2000**, *12*, 1057–1063.
19. Schulman, J.; Moritz, P.; Levine, S.; Jordan, M.I.; Abbeel, P. High-dimensional continuous control using generalized advantage estimation. In Proceedings of the 4th International Conference on Learning Representations, ICLR 2016, Conference Track Proceedings, San Juan, Puerto Rico, 2–4 May 2016.
20. Ueno, T.; Maeda, S.I.; Kawanabe, M.; Ishii, S. Generalized TD learning. *J. Mach. Learn. Res.* **2011**, *12*, 1977–2020.
21. Jang, B.; Kim, M.; Harerimana, G.; Kim, J.W. Q-Learning Algorithms: A Comprehensive Classification and Applications. *IEEE Access* **2019**, *7*, 133653–133667. [CrossRef]
22. Baxter, L.A.; Puterman, M.L. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*; Wiley Publishing: Hoboken, NJ, USA, 1995; Volume 37. [CrossRef]
23. Farquhar, G.; Rocktäschel, T.; Igl, M.; Whiteson, S. TreeqN and ATreEC: Differentiable tree-structured models for deep reinforcement learning. In Proceedings of the 6th International Conference on Learning Representations, ICLR 2018, Conference Track Proceedings, Vancouver, BC, Canada, 30 April–3 May 2018.
24. Schrittwieser, J.; Antonoglou, I.; Hubert, T.; Simonyan, K.; Sifre, L.; Schmitt, S.; Guez, A.; Lockhart, E.; Hassabis, D.; Graepel, T.; et al. Mastering Atari, Go, chess and shogi by planning with a learned model. *Nature* **2020**, *588*, 604–609. [CrossRef]
25. Ginsberg, M.L. GIB: Imperfect Information in a Computationally Challenging Game. *J. Artif. Intell. Res.* **2001**, *14*, 303–358. [CrossRef]
26. Frank, I.; Basin, D. Search in games with incomplete information: A case study using Bridge card play. *Artif. Intell.* **1998**, *100*, 87–123. [CrossRef]
27. Srivastava, N.; Hinton, G.; Krizhevsky, A.; Sutskever, I.; Salakhutdinov, R. Dropout: A simple way to prevent neural networks from overfitting. *J. Mach. Learn. Res.* **2014**, *15*, 1929–1958.
28. Mizukami, N.; Tsuruoka, Y. Building a computer Mahjong player based on Monte Carlo simulation and opponent models. In Proceedings of the 2015 IEEE Conference on Computational Intelligence and Games, CIG 2015, Tainan, Taiwan, 31 August–2 September 2015. [CrossRef]
29. van Rijn, J.N. Mahjong solitaire computing the number of unique and solvable arrangements. *Mathematics* **2011**, *51*, 32–37.
30. Schmidhuber, J. Deep Learning in Neural Networks: An Overview. *Neural Netw.* **2015**, *61*, 85–117. [CrossRef]
31. 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. *arXiv* **2020**, arXiv:2003.13590.
32. Gao, S.; Li, S. Bloody Mahjong playing strategy based on the integration of deep learning and XGBoost. *CAAI Trans. Intell. Technol.* **2022**, *7*, 95–106. [CrossRef]
33. Liu, K.; Bellet, A.; Sha, F. Similarity learning for high-dimensional sparse data. *arXiv* **2015**, arXiv:1411.2374.
34. Georganos, S.; Grippa, T.; Vanhuysse, S.; Lennert, M.; Shimoni, M.; Wolff, E. Very High Resolution Object-Based Land Use–Land Cover Urban Classification Using Extreme Gradient Boosting. *IEEE Geosci. Remote. Sens. Lett.* **2018**, *15*, 607–611. [CrossRef]
35. Zheng, Y.; Li, S. A Review of Mahjong AI Research. In Proceedings of the 2020 2nd International Conference on Robotics, Intelligent Control and Artificial Intelligence (RICAI '20), Shanghai, China, 17–19 October 2020; Association for Computing Machinery: New York, NY, USA, 2020; pp. 345–349. [CrossRef]
36. Krizhevsky, A.; Sutskever, I.; Hinton, G.E. Imagenet classification with deep convolutional neural networks. *Commun. ACM* **2017**, *60*, 84–90. [CrossRef]
37. Tsantekidis, A.; Passalis, N.; Tefas, A. Recurrent neural networks. *Deep. Learn. Robot. Percept. Cogn.* **2022**, *30*, 101–115. [CrossRef]
38. Browne, C.B.; Powley, E.; Whitehouse, D.; Lucas, S.M.; Cowling, P.I.; Rohlfshagen, P.; Tavener, S.; Perez, D.; Samothrakis, S.; Colton, S. A Survey of Monte Carlo Tree Search Methods. *IEEE Trans. Comput. Intell. AI Games* **2012**, *4*, 1–43. [CrossRef]
39. Gonçalves, R.A.; Almeida, C.P.; Pozo, A. Upper confidence bound (UCB) algorithms for adaptive operator selection in MOEA/D. In *Evolutionary Multi-Criterion Optimization, Proceedings of the 8th International Conference, EMO 2015, Guimaraes, Portugal, 29 March–1 April 2015*; Springer: Berlin/Heidelberg, Germany, 2015; Volume 9018, p. 9018. [CrossRef]
40. Tuglu, N.; Kuş, S. q-Bernoulli matrices and their some properties. *Gazi Univ. J. Sci.* **2015**, *28*, 269–273.
41. Rolnick, D.; Ahuja, A.; Schwarz, J.; Lillicrap, T.P.; Wayne, G. Experience replay for continual learning. *Adv. Neural Inf. Process. Syst.* **2019**, *32*, 78–89.
42. Ioffe, S.; Szegedy, C. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In Proceedings of the 32nd International Conference on Machine Learning, ICML 2015, Lille, France, 6 July–11 July 2015; Volume 1.
43. Gao, S.; Okuya, F.; Kawahara, Y.; Tsuruoka, Y. Supervised learning of imperfect information data in the game of mahjong via deep convolutional neural networks. *Inf. Process. Soc. Jpn.* **2018**, *4*, 56–78.
44. Wang, J.; Xu, X.H. Computer Game: The Frontier of Artificial Intelligence: National University Student Computer Game Contest. *Comput. Educ.* **2012**, *163*, 14–18.
45. Yoon, J.; Jeong, B.; Kim, M.; Lee, C. An information entropy and latent Dirichlet allocation approach to noise patent filtering. *Adv. Eng. Informatics* **2021**, *47*, 101243. [CrossRef]
46. Sutskever, I.; Martens, J.; Dahl, G.; Hinton, G. On the importance of initialization and momentum in deep learning. In Proceedings of the 30th International Conference on Machine Learning, ICML 2013, Atlanta, GA, USA, 16–21 June 2013. PART 3.

47. Mirsoleimani, S.A.; Plaat, A.; Herik, J.V.D.; Vermaseren, J. An analysis of virtual loss in parallel MCTS. ICAART 2017. In Proceedings of the 9th International Conference on Agents and Artificial Intelligence, Lisbon, Portugal, 22–24 February 2013; Volume 2. [CrossRef]
48. Dean, J.; Corrado, G.S.; Monga, R.; Chen, K.; Devin, M.; Le, Q.V.; Mao, M.Z.; Ranzato, M.A.; Senior, A.; Tucker, P.; et al. Large scale distributed deep networks. *Adv. Neural Inf. Process. Syst.* **2012**, *2*, 25–28.