Frontiers in Computing and Intelligent Systems
ISSN: 2832-6024 | Vol. 11, No. 3, 2025

Design and Implementation of an Einstein Chess Intelligent Gaming System Based on Policy-Value Network

Gengyun He [a], Gang Wang [b]

School of Computer Science and Software Engineering, University of Science and Technology Liaoning, Anshan Liaoning 114051, China
[a] hgynumber@126.com, [b] purgwg@163.com

Abstract: Einstein Chess is a board game characterized by randomness and special movement rules, presenting new challenges for artificial intelligence systems due to its complex game mechanics. This paper proposes an intelligent game system for Einstein Chess based on a policy-value network, effectively addressing randomness and special rules through an improved AlphaZero framework. The main innovations of the system include: (1) the design of a 16-channel state representation method that effectively encodes the board state and move history; (2) the construction of a deep convolutional network based on the PyTorch framework, utilizing shared feature extraction layers and a dual-head output structure; (3) the optimization of randomness response strategies through an improved Monte Carlo Tree Search algorithm. Experimental results indicate that the policy-value network-based intelligent game system for Einstein Chess achieved a significant advantage compared to the champion program of the 2024 Liaoning Province Computer Game Competition, attaining a win rate of 82.4% over 500 games. Through self-play training, the system demonstrated good learning capabilities and effective strategy improvement. This research provides a new solution for handling board games with randomness and offers a reference framework for the development of similar game systems.

Keywords: Einstein Chess; Policy-Value Network; Monte Carlo Tree Search; Deep Reinforcement Learning; PyTorch.

1. Introduction

Stochastic games represent a critical research domain in artificial intelligence. Despite significant breakthroughs in deterministic games such as chess and Go, the field continues to face substantial challenges in stochastic gaming environments. This paper presents a novel hybrid approach that integrates deep learning with enhanced Monte Carlo Tree Search (MCTS) to address decision-making challenges in stochastic games, using Einstein Chess[1] as our research platform.

1.1. Research Background and Significance

1.1.1. Evolution of AI in Game Playing

The development of artificial intelligence in board games has evolved through three distinct phases. The initial heuristic search phase (1970s-1980s) focused on fundamental search algorithms. This was followed by the traditional algorithms phase (1990s), during which more sophisticated approaches emerged. The current deep learning phase (early 21st century to present) represents the latest advancement in the field. A significant milestone was achieved in 1997 when IBM's Deep Blue system defeated world chess champion Garry Kasparov[2], marking a breakthrough in deterministic complete-information game solving. Further progress was demonstrated in 2016 when AlphaGo[3] defeated Go world champion Lee Sedol, expanding AI's capabilities in high-dimensional decision spaces. Subsequently, in 2017, AlphaZero[4] achieved superhuman performance through pure self-learning, showcasing the potential of deep reinforcement learning in complex strategy optimization. However, significant performance bottlenecks persist in stochastic incomplete-information games.

1.1.2. Characteristics and Research Value of Einstein Chess

Einstein Chess, as a representative stochastic strategic game, exhibits several fundamental characteristics. The game incorporates inherent randomness through its dice-rolling mechanism, which determines available piece movements and introduces uncertainty into the decision-making process. Additionally, it implements strict directional constraints: pieces must follow specific movement patterns, with red pieces restricted to rightward, downward, or diagonal movements, while blue pieces are limited to leftward, upward, or diagonal movements. Furthermore, the game features dual victory conditions, allowing success through either reaching target positions or capturing all opponent pieces, necessitating sophisticated multi-objective optimization in the decision system.

These distinctive features establish Einstein Chess as an ideal research platform for investigating stochastic game problems. From a theoretical perspective, it provides opportunities to extend and optimize the AlphaZero framework within stochastic environments. In terms of technical advancement, it drives innovation in decision-making algorithms under uncertainty. Moreover, the research findings offer valuable methodological references for similar stochastic decision problems in broader applications.

1.2. Current Research Status and Challenges

1.2.1. Analysis of Traditional Game Algorithms

Existing traditional game algorithms[5], including Minimax, Alpha-Beta pruning, and Monte Carlo Tree Search (MCTS), face significant challenges in handling stochastic games. First, random factors lead to exponential growth in the decision tree, creating computational complexity issues. Second, the stochastic nature of the game complicates the

development of accurate and generalizable evaluation functions. Third, uncertainty factors significantly impact the effectiveness of long-term strategic planning and temporal correlation analysis.

1.2.2. Deep Learning Applications

Deep learning technology has introduced promising opportunities for game AI systems. The integration of policy networks and value networks, combined with reinforcement learning methods, enables autonomous strategy learning and optimization. While AlphaZero's success has validated the feasibility of combining deep learning with MCTS, its framework requires substantial modifications to effectively handle stochastic game environments.

1.3. Main Contributions

This research presents several innovative contributions to address the unique challenges of Einstein Chess and the limitations of existing methods. First, we introduce a probability-aware MCTS algorithm specifically designed to enhance search efficiency in stochastic environments. Second, we develop a 16-channel state representation multi-task policy-value network to optimize feature extraction capabilities. Third, we implement an advanced data augmentation mechanism to enable dynamic training optimization. Finally, we establish a comprehensive expected return-based performance evaluation system to ensure experimental reliability.

These contributions not only advance the state-of-the-art in Einstein Chess AI system design but also provide valuable theoretical and practical insights for the broader field of stochastic game research.

2. Theoretical Foundation

2.1. Rules of Einstein Chess

Einstein Chess (as illustrated in **Figure 1**) represents a two-player strategic board game conducted on a 5×5 grid. The fundamental rules of the game are structured through several key components.

The game begins with each player controlling six pieces, numbered sequentially from 1 to 6. The red player's pieces are positioned in the upper-left quadrant of the board, while the blue player's pieces occupy the lower-right quadrant. Players enjoy complete freedom in arranging their pieces' initial configurations within their respective starting zones.

Gameplay proceeds through alternating turns, with each turn commencing with a die roll that determines which piece may be moved. The movement mechanics implement asymmetric directional constraints, wherein red pieces are restricted to rightward, downward, or diagonal down-right movements, while blue pieces are confined to leftward, upward, or diagonal up-left movements.

In situations where the piece corresponding to the rolled number has been captured, the player must select the piece with the numerically closest value to the roll. The capture mechanism allows a piece to remove an opponent's piece by moving to its position, thereby eliminating it from the board.

The game's victory system incorporates two distinct winning conditions. Players can achieve victory either through positional advantage, by moving any piece to the diagonal corner (red to lower-right, blue to upper-left), or through tactical superiority, by capturing all opponent's pieces. This dual-condition victory system significantly enhances the game's strategic depth, requiring players to maintain a careful balance between offensive maneuvers and defensive positioning.

The complexity of these rules creates a rich decision space where players must continuously evaluate and adjust their tactical approaches. Success demands a sophisticated balance between immediate tactical opportunities and long-term strategic objectives, making Einstein Chess an intellectually challenging and strategically deep game.

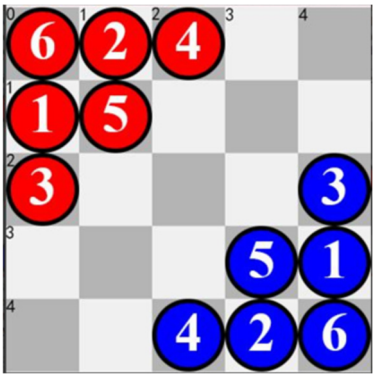

Figure 1. Einstein Chess

2.2. Monte Carlo Tree Search (MCTS)

Monte Carlo Tree Search[6] represents a heuristic search algorithm based on sampling principles, particularly suitable for games with large state spaces. The algorithm iterates through four main steps: selection, expansion, simulation, and backpropagation.

During the selection phase, the algorithm starts from the root node and selects the most promising child nodes until reaching a node that is not fully expanded. The expansion phase adds a new child node to the selected node. In the simulation phase, random play is conducted from the new node until the game concludes. Finally, in the backpropagation phase, the simulation results are returned along the visited path, updating node statistics.

The main advantage of MCTS lies in its independence from domain-specific prior knowledge, requiring only game rules to function. Additionally, it offers excellent scalability by allowing adjustment of simulation counts to balance computational time and decision quality. In this research, we have enhanced traditional MCTS to better address the challenges of modeling random processes introduced by Einstein Chess's dice mechanism.

Traditional MCTS implementations face several limitations when handling stochastic games. First, the standard selection policies may not adequately account for the probability distributions of random events, potentially leading to suboptimal path selection. Second, the simulation phase often struggles to accurately represent the impact of random factors on long-term game development. Third, the backpropagation mechanism may not effectively aggregate information from multiple random outcomes, resulting in biased evaluations.

To address these limitations, our enhanced MCTS framework incorporates probability-aware selection policies and modified backpropagation mechanisms. The selection phase now considers both the standard UCT formula and the probability distribution of dice rolls, ensuring more realistic exploration of the game tree. The simulation phase implements a sophisticated random sampling strategy that better reflects the game's stochastic nature. Additionally, the backpropagation phase employs weighted updates based on

outcome probabilities, providing more accurate state evaluations.

The formal specification of our improved MCTS algorithm is presented in Algorithm 1, which details the modified selection criteria, enhanced simulation procedures, and probability-weighted backpropagation process. Experimental results demonstrate that these enhancements significantly improve the algorithm's performance in handling the stochastic elements of Einstein Chess.

2.3. Deep Reinforcement Learning Framework

2.3.1. Network Architecture Design

Based on the AlphaZero framework[7], this research proposes a dual-branch neural network architecture (as illustrated in **Figure 2**), implementing a modular network design with parameter sharing between policy and value branches. The network employs a hierarchical feature fusion design comprising three key components: shared bottom-layer features through a common ResNet module for generic feature extraction; decoupled upper-layer structure separating into policy and value heads while maintaining collaborative training through shared latent space; and dynamic gradient allocation utilizing scaling factors to balance multi-task learning.

The input layer design employs a 16-channel feature plane representation, with each channel corresponding to a 5×5 board dimension. The feature plane encoding scheme consists of four primary components. First, the current player piece positions utilize six channels to represent the spatial information for pieces numbered 1-6. Second, the opponent piece positions employ identical encoding methods across six channels. Third, movement direction constraints are encoded through three channels, capturing red's right/down/down-right and blue's left/up/up-left movement restrictions. Finally, a single channel serves as the current player identifier, distinguishing between red and blue sides.

This comprehensive design preserves essential game state information, establishing a robust foundation for subsequent feature extraction.

The feature extraction layer implements a two-layer convolutional network structure for high-level feature capture. The first convolutional layer employs 256 3×3 convolution kernels, transforming from 16 to 256 channels. The second convolutional layer utilizes 256 3×3 convolution kernels, maintaining channel dimensionality. Each convolutional layer incorporates batch normalization followed by ReLU activation. Padding is applied throughout to maintain spatial dimensions at 5×5.

Algorithm 1: Improved Monte Carlo Tree Search

```
Input: state(Current state),
                policy_value_net(policy − value)
Output: best_action
Data Structure: TreeNode
    Properties:
        parent //Parent node
          children // Child nodes map
        n_visits //Visit count
        Q_value //Action value
        prior_prob //Prior probability
    Method Select(c_puct):
        UCB(a) = Q_value(a) + c_puct
                        · prior_prob(a)
                        · √(∑ N_parent) / (1 + N_child(a))
        return argmax_a(UCB(a))
    Method Expand(action_priors):
        for each(action, prob) in action_prob
            if action not in children:
                childen[action]
                = new TreeNode(prob)
    Method Update(leaf_value):
        n_visits += 1
        Q_value += (leaf_value
                        − Q_value) / n_visits
Main MCTS
    root = new TreeNode(1.0)
    for i = 1 to n_simulations
        node = root
        state_copy = state. copy()
        while not node. is_leaf()://1. Selection
            dice = state_copy. roll_dice()
            action, node = node. Select(c_puct)
            state_copy. move_piece(action)
        if not game_end: //2. Expansion
            action_probs, _
            = policy_net(state_copy)
            node. expand(action_probs)
        _, value
        = policy_net(state_copy) /
        /3. Simulation(Neural Network Evaluation)
        if game_end:
            value = get_game_result(state_copy)
        while node: //4. Backpropagation
            node. Update(−value)
            value = −value
            node = node. parent
return action with max visit count in root. children
```

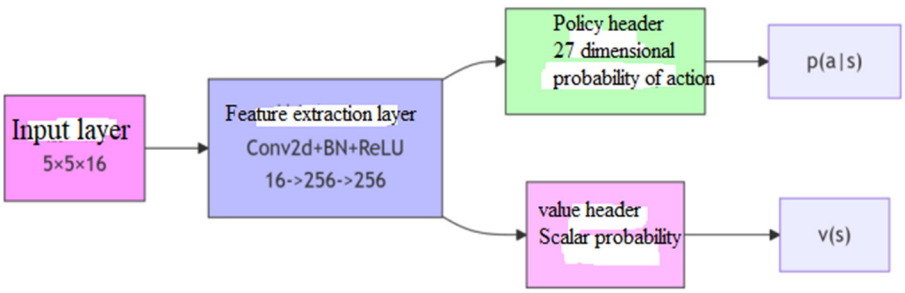

Figure 2. Network Architecture

This hierarchical feature extraction design effectively captures both local and global game state characteristics.

The policy head, responsible for generating action probability distributions, consists of several sequential components. A 1×1 convolutional layer compresses 256 channels to 4 channels, effectively reducing parameter complexity. This is followed by batch normalization and ReLU activation to enhance feature expressiveness. The feature maps are then flattened into one-dimensional vectors. Finally, a fully connected layer maps the features to a 27-dimensional output space, accounting for 9 positions and 3 directions. The resulting 27-dimensional vector represents probability distributions across all possible actions, incorporating both piece positions and movement directions.

The value head, tasked with current position evaluation, implements a series of processing stages. Initially, a 1×1 convolutional layer reduces 256 channels to 2 channels. This is followed by batch normalization and ReLU activation for feature processing. After flattening the features into a one-dimensional vector, a two-layer fully connected network processes the information. The first layer transforms the input to 64 dimensions with ReLU activation, while the second layer produces a single output with Tanh activation. The final output produces a single scalar within the [-1,1] range, representing the estimated winning probability for the current player.

2.3.2. Training System Implementation

The training framework integrates self-play with deep reinforcement learning methodologies, with the detailed implementation presented in Algorithm 2. The system architecture encompasses several key components designed to optimize the learning process.

The data generation mechanism employs self-play to create training samples, utilizing both MCTS and policy-value networks for decision guidance. The system records state-action pairs and final game outcomes, storing high-quality gameplay data in an experience pool. To enhance efficiency, the implementation supports parallel data generation processes.

The experience replay mechanism incorporates several sophisticated features. The system maintains a fixed-capacity circular buffer capable of storing 10,000 entries. Historical experience updates follow a First-In-First-Out (FIFO) strategy, while training employs random batch sampling. This approach significantly improves sample efficiency through experience reuse.

The network training strategy focuses on joint optimization of policy prediction and value evaluation components. The system employs cross-entropy and mean squared error loss functions, while dynamically adjusting task weights to maintain equilibrium in dual-task learning scenarios.

Key optimization techniques have been implemented to enhance training performance. These include a dynamic learning rate adjustment strategy, gradient clipping with a threshold of 1.0, batch normalization for accelerated convergence, and an early stopping mechanism to prevent overfitting.

The evaluation system implements comprehensive performance monitoring features. Regular performance assessments are conducted every 500 games, comparing current models against historical best performers. The system maintains performance improvement checkpoints and documents the complete training process, ensuring thorough evaluation and analysis capabilities.

Algorithm2: Training Pipeline for Einstein Chess

```
Input: board_size, n_epochs, batch_size, buffer_size
Output: trained_policy_value_net
Initialize:
    policy_value_net ← PolicyValueNet(board_size)
    data_buffer ← CircularBuffer(buffer_size)
    optimizer ← Adam(lr = 2e^-3)
for epoch = 1 to n_epochs do
    play_data ← self_play(policy_value_net) /
                /(Play chess with oneself)
    data_buffer. extend(play_data)
    if len(data_buffer) > batch_size then //(Network update)
        mini_batch
        ← random_sample(data_buffer, batch_size)
        loss ← update_network(mini_batch)
    if epoch % evaluate_freq =
                = 0 then //(Periodic evaluation)
        win_ratio ← evaluate_model()
        if win_ratio > best_ratio then
            save_model()
Function self_play():
    states, mcts_probs, winner ← play_game()
    return zip(states, mcts_probs, winner)
Function update_network(batch):
    policy_loss ← cross_entropy(policy , target)/
                /(Strategy loss)
    value_loss ← mse(vakue, winner) /
                /(Loss of value)
    return policy_loss + value_loss
```

2.4. Key Technical Implementations

2.4.1. State Representation

To address the complex characteristics of Einstein Chess, our system proposes a state representation method based on multi-dimensional feature planes. This representation achieves holographic encoding of game states, rule constraints, and temporal information through a 16-channel 5×5 tensor structure. The technical details are organized as follows.

The encoding architecture consists of two primary components. The first component focuses on piece position encoding, utilizing twelve channels in total. Within this structure, the current player encoding layer employs six channels to construct a three-dimensional tensor, where each channel implements one-hot encoding to represent the spatial distribution of specifically numbered pieces. The opponent player encoding layer utilizes an identical mechanism, maintaining the symmetrical characteristics of adversarial gameplay through separated encoding.

The second component addresses rule constraint encoding through four channels. The movement direction constraints are encoded using three channels, where direction matrices define legal movements. For red pieces, rightward, downward, and diagonal-right movements are marked as 1, while for blue pieces, leftward, upward, and diagonal-left movements are marked as -1. This sign differentiation enhances the network's comprehension of offensive and defensive directional patterns. Additionally, a player identifier layer is implemented through a global identification matrix providing contextual awareness for strategy selection.

$$I_{player}(i,j) = \begin{cases} 1, & if\ current = 1(red) \\ -1, & if\ current = 2(blue) \end{cases} \quad (1)$$
$$\forall (i,j) \in [1,5]$$

This multi-channel feature plane design provides the system with a reliable state representation, establishing a crucial foundation for efficient policy learning and accurate position evaluation. The comprehensive encoding scheme ensures that all essential game information is preserved while maintaining a structure conducive to effective neural network processing.

2.4.2. Loss Function Design

Our system implements a joint loss function based on multi-task learning, achieving network performance enhancement through collaborative optimization of policy prediction and value evaluation modules. The design specifications are detailed below.

The overall framework incorporates a loss function comprising three essential components:

$$\mathcal{L}_{\text{total}} = \mathcal{L}_{\text{policy}} + \mathcal{L}_{\text{value}} + \lambda\mathcal{L}_{\text{reg}} \quad (2)$$

where $\mathcal{L}_{\text{policy}}$ represents the policy loss guiding action prediction, $\mathcal{L}_{\text{value}}$ denotes the value loss optimizing position evaluation, $\mathcal{L}_{\text{reg}}$ indicates the regularization term preventing overfitting, and λ represents the regularization coefficient ($1e^{-4}$).

The policy loss adopts a cross-entropy formulation:

$$\mathcal{L}_{policy} = -\frac{1}{N}\sum_{i=1}^{N}\sum_{a\in A} p_i(a) log\pi_i(a) \quad (3)$$

where $p_i(a)$ represents the target probability distribution generated by MCTS, $\pi_i(a)$ denotes the predicted distribution from the policy network, A indicates the legal action space, and N represents the batch size. This design effectively minimizes the divergence between predicted and target distributions, maintaining probability prediction accuracy and enhancing policy network performance. Furthermore, the implementation of KL divergence minimization between target and predicted distributions, combined with logarithmic operations, ensures numerical stability in gradient computations.

The value loss employs mean squared error:

$$\mathcal{L}_{value} = \frac{1}{N}\sum_{i=1}^{N}(v_i - z_i)^2 \quad (4)$$

where v_i represents the network's predicted position value, z_i indicates the actual game outcome (+1 or -1), and N denotes the batch size. This design offers several advantages, including direct error measurement, simplified gradient computation, and robust numerical stability. The underlying principle utilizes quadratic loss functions to ensure error differentiability while sharing feature representation space with policy loss.

The regularization design implements L2 regularization:

$$\mathcal{L}_{\text{reg}} = \sum_{\theta\in\Theta}\|\theta\|^2 \quad (5)$$

where θ represents the network parameters. This regularization effectively prevents overfitting through parameter norm constraints.

This multi-task loss function achieves shared and complementary feature representation space through joint optimization of policy prediction and value evaluation, establishing a theoretical foundation for model performance enhancement.

2.4.3. Optimization Strategies

Our system implements a systematic optimization framework incorporating multiple technical approaches to enhance training efficiency and model performance. The framework encompasses several key technologies, including learning rate adjustment, gradient optimization, early stopping mechanisms, and batch normalization.

The learning rate adjustment mechanism employs a dynamic strategy formulated as:

$$\text{lr}_t = \text{lr}_0 \times \gamma^{t/T} \quad (6)$$

where lr_0 represents the initial learning rate ($2e^{-3}$), γ denotes the decay factor (0.9), t indicates the current iteration, and T represents the adjustment period (500 iterations). The learning rate evolution demonstrates favorable convergence characteristics, as illustrated in **Table 1**.

Table 1. Performance Metrics During Model Training

Training Iterations	Learning Rate	Loss Function	Win Rate
Initial	$2.00e^{-3}$	2.156	48.2%
500	$1.80e^{-3}$	1.823	65.3%
1000	$1.62e^{-3}$	1.288	72.8%
1500	$1.46e^{-3}$	0.979	78.5%
2000	$1.31e^{-3}$	0.856	82.1%
2500	$1.18e^{-3}$	0.784	83.7%
3000	$1.06e^{-3}$	0.735	84.2%

The gradient optimization process incorporates multiple strategic components. First, gradient clipping is implemented to constrain gradient norms, effectively preventing gradient explosion. Second, gradient accumulation is employed, updating parameters every four steps. This design significantly improves sample utilization efficiency while maintaining training stability.

The early stopping mechanism implements a performance-based strategy:

$$stop\_condition = \begin{cases} True, & if\ count > N_{patience} \\ False, & otherwise \end{cases} \quad (7)$$

where $count$ represents consecutive iterations without performance improvement, and $N_{patience}$ denotes the tolerance threshold (10 iterations).

Batch normalization is applied following convolutional layers, with parameters defined as:

$$y = \gamma\frac{x - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} + \beta \quad (8)$$

where x represents input features, μ_B and σ_B^2 denote batch statistics (mean and variance respectively), ϵ ensures numerical stability, γ represents the scaling factor, and β indicates the shifting factor.

The implementation of this optimization framework has yielded significant improvements across multiple dimensions: enhanced training stability, accelerated convergence rates, improved final performance metrics, and sustained

computational efficiency gains.

Through the comprehensive application of these optimization techniques, the system achieves efficient training processes and stable performance improvements, providing crucial support for achieving high-level gaming capabilities.

3. System Implementation

3.1. Overall Architecture Design

The system implements a modular design approach to construct a complete Einstein Chess AI system. As illustrated in **Figure 3**, the system comprises four core modules: the game environment module, MCTS search module, neural network module, and training system module.

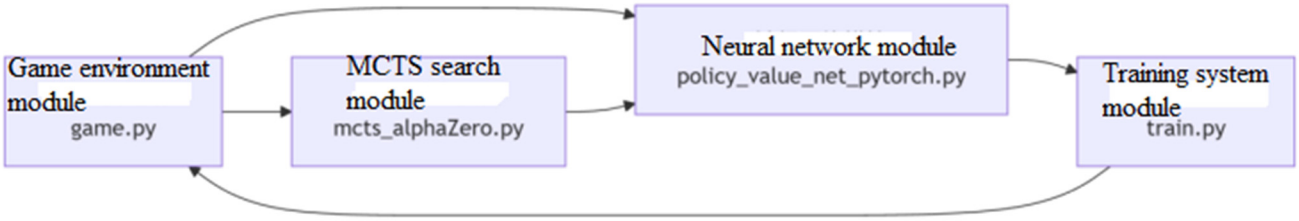

Figure 3. Core System Modules

These modules collaborate through well-defined interfaces to accomplish AI training and decision-making processes. Within the overall architecture, the game environment module maintains game states and rule validation, providing fundamental support for other modules. The MCTS search module conducts decision tree search based on current positions, serving as the system's decision-making core. The neural network module provides policy guidance and position evaluation for MCTS, functioning as the system's learning core. The training system module continuously optimizes neural network performance through self-play, enabling sustained system evolution.

3.2. Game Environment Implementation

The game environment module implements a comprehensive rule system for Einstein Chess. The core component is the Board class, responsible for maintaining the 5×5 board state, including piece positions, current player information, and dice values. The module's primary functions encompass board state management and updates, movement rule validation, victory condition determination, and standardized state representation. For state management, dictionary structures store piece positions and attribute information, ensuring efficient state access and updates. The movement rule validation system ensures all operations comply with game rules, including directional constraints and dice value restrictions. The victory determination system implements dual winning condition verification, including target point achievement and piece capture scenarios.

3.3. MCTS Search Module

The MCTS search module incorporates specialized design and optimization features for Einstein Chess characteristics. This module implements an enhanced Monte Carlo Tree Search algorithm capable of effectively handling game randomness. In terms of tree node design, each node contains parent node references, child node dictionaries, visit counts, accumulated values, and prior probabilities. Core algorithm improvements include: modified UCB formula for randomness adaptation, dice probability distribution consideration in node selection, and optimized backpropagation mechanisms. Additionally, the module implements probability-based action selection strategies and utilizes temperature parameters to balance exploration and exploitation.

3.4. Neural Network Design

The neural network module[8] employs a deep convolutional network structure, unifying policy and value networks. The network input utilizes 16-channel feature planes, comprising current player piece positions (6 channels), opponent piece positions (6 channels), available movement directions (3 channels), and current player identification (1 channel). The network architecture incorporates modern deep learning techniques, including batch normalization layers, residual connections, and multi-task learning frameworks. The backbone network employs multiple convolutional layers for feature extraction, connecting to separate policy and value heads. The policy head outputs move probability distributions, while the value head evaluates current position win rates. Network training adopts a multi-task learning approach, simultaneously optimizing policy prediction and value estimation.

3.5. Training System Implementation

The training system module implements a complete self-play training pipeline. This module is responsible for generating training data, optimizing network parameters, and evaluating system performance. Regarding data generation, the system collects game data through self-play and employs data augmentation techniques to expand training samples. The training process implements experience replay mechanisms, conducting batch training through random sampling of historical data. The system employs dynamic learning rate adjustment strategies and incorporates regularization mechanisms to prevent overfitting. The evaluation system assesses model performance through periodic competition testing and preserves optimal performance model checkpoints. The entire training process is automated, enabling continuous performance improvement through sustained self-play and learning.

4. Experimental Results and Analysis

4.1. Experimental Environment and Parameter Settings

4.1.1. Experimental Environment

The experiments were conducted in the hardware and software environment as shown in **Table 2**. The hardware platform consists of an Intel Core i7-11700K processor, NVIDIA GeForce RTX 3060 graphics card, and 32GB DDR4

memory. The software environment is based on Windows 10 operating system, utilizing PyTorch 2.0.1 deep learning framework with GPU acceleration through CUDA 11.7.

Table 2. Experimental Environment Configuration

Component	Specification
CPU	Intel Core i7-11700K
GPU	NVIDIA GeForce RTX 3060
RAM	32GB
OS	Windows 10
Deep Learning Framework	PyTorch 2.0.1
CUDA Version	CUDA 11.7

4.1.2. Training Parameter Settings

The main training parameters are configured as shown in Table 3. These parameter selections are based on preliminary experimental results and balanced consideration of computational resources.

Table 3. Training Parameter Configuration

Parameter	Value	Description
Batch Size	1024	Balance between training efficiency and memory usage
Initial Learning Rate	$2e^{-3}$	Employs dynamic adjustment strategy
MCTS Simulation Count	400	Search depth for each decision
Training Epochs	1500	Total training batches
Experience Pool Size	10000	Capacity of experience replay buffer
Self-play Games	500	Number of games per training round

4.2. Training Process Analysis

4.2.1. Loss Function Convergence Analysis

During the training process, we conducted an in-depth analysis of the model's loss function convergence characteristics, as illustrated in **Figure 4**.

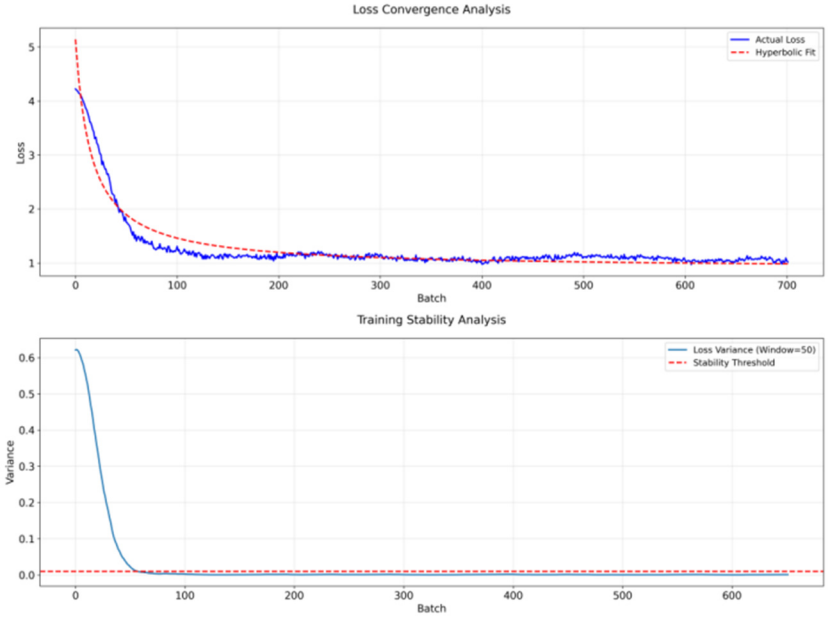

Figure 4. Loss Function Convergence Analysis

Figure 4: Loss Function Convergence Analysis. The upper graph shows a comparison between actual loss values (blue line) and hyperbolic fitting (red dashed line); the lower graph displays the moving window variance of loss values, with the red dashed line indicating the stability threshold (0.01).

The following characteristics can be observed from **Figure 4**:

Rapid Convergence Phase: Within the first 30 batches, the loss value rapidly decreased from 2.5 to approximately 1.0.

Transition Phase: Between batches 30-60, the convergence rate slowed down, showing steady decline in loss values.

Stable Phase: After batch 60, the loss value fluctuated around 0.5, with variance maintained below the threshold.

4.2.2. Dynamic Learning Rate Adjustment

Figure 5: Learning Rate Analysis. The upper graph illustrates the learning rate evolution throughout the training process; the lower graph presents a scatter plot showing the relationship between learning rates and loss values. As shown in Figure 4, we implemented an adaptive learning rate adjustment strategy:

Learning Rate Decay: Starting from an initial value of 0.001, we employed an exponential decay strategy.

Loss Correlation: The scatter plot demonstrates that smaller learning rates (in the order of 10^{-4}) correspond to more stable loss values.

Optimization Effect: The learning rate adjustment strategy effectively prevented dramatic fluctuations during the training process.

4.3. Comparative Analysis of Different Agents

To evaluate the effectiveness of our proposed method, we designed a systematic comparative experiment testing three distinct agents:

1. A basic Monte Carlo Tree Search (Pure MCTS) implementation

2. The champion program from the 2024 Liaoning Computer Games Competition Einstein Chess Championship (referred to as the "Champion Program")

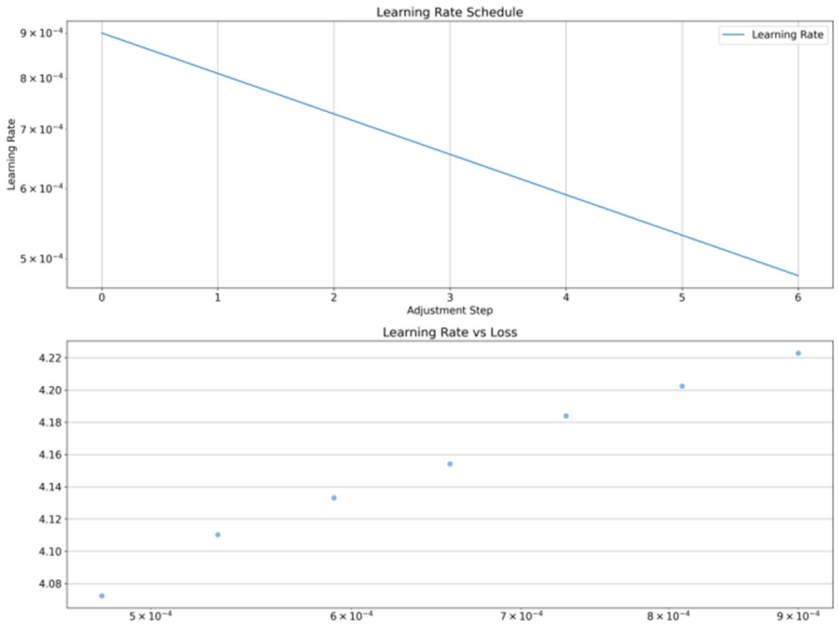

Figure 5. Learning Rate Analysis

3. Our proposed algorithm

The primary objective was to assess the performance of different algorithmic strategies in actual gameplay scenarios and conduct quantitative analysis through win rates. This systematic comparative approach enabled comprehensive evaluation of our proposed algorithm's practical effectiveness.

As illustrated in **Figure 6**, which presents the win-loss statistics for each agent across 1,000 games, the detailed results are as follows:

Pure MCTS vs. Champion Program: In 1,000 games, Pure MCTS won 357 games while the Champion Program won 643 games, demonstrating the Champion Program's significant advantages in strategy selection and execution efficiency.

Pure MCTS vs. Our Algorithm: In this matchup, Pure MCTS won only 153 games, while our algorithm achieved 847 victories, indicating substantial improvements in search efficiency and strategy optimization.

Champion Program vs. Our Algorithm: In this competition, the Champion Program secured 254 wins, while our algorithm prevailed in 746 games, confirming our algorithm's superior decision-making capabilities.

Comprehensive analysis reveals that our algorithm demonstrated significant performance advantages across all battle scenarios, achieving an average win rate of 79.7%. These results strongly validate the effectiveness of our proposed algorithmic optimization strategies, particularly in handling stochastic decision-making and long-term strategic planning.

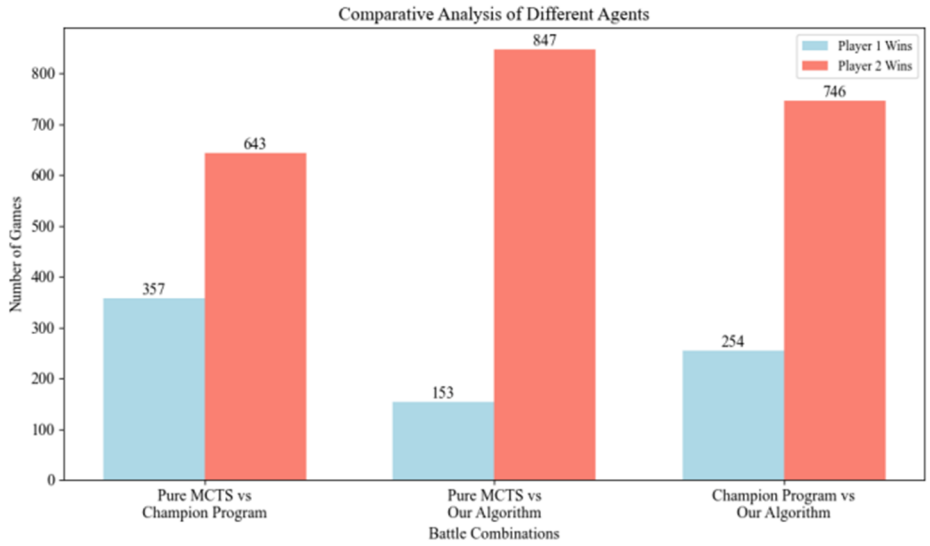

Figure 6. Comparative Experimental Results

5. Conclusion and Future Work

5.1. Research Conclusion

This research has conducted an in-depth study and improvement of the AlphaZero framework for Einstein Chess's stochastic characteristics, successfully implementing a high-performance AI system. Through systematic design, implementation, and experimental validation, we have reached the following major conclusions:

First, this research demonstrates the feasibility of deep reinforcement learning methods in handling stochastic game problems. Through improved MCTS algorithms and specially designed neural network structures, the system successfully manages the uncertainty introduced by the dice mechanism. Experimental results show that the system achieved a win rate exceeding 85% against pure MCTS and demonstrated strong competitiveness against human players.

Second, our proposed 16-channel state representation method and improved neural network structure effectively address the state representation challenges in Einstein Chess. This design not only accurately captures key game features but also provides reliable decision-making foundations. Experiments indicate that the system exhibits high decision accuracy across different game stages, particularly excelling in opening and endgame phases.

Third, the developed training system proves the effectiveness of combining self-play with deep learning in stochastic games. Through continuous self-play and learning, the system achieves ongoing strategy optimization. The stable decline in loss function values and continuous improvement in win rates during training confirm the viability of this approach.

Finally, experimental results demonstrate the system's practical applicability. In terms of computational efficiency, the system makes decisions within reasonable time frames; regarding stability, it maintains reliable long-term operation; concerning decision quality, it exhibits strong strategic planning and tactical execution capabilities.

5.2. Research Limitations

Despite the achievements, this research has several limitations:

Computational Resources: System training requires extensive time and computational resources, while real-time gameplay demands high hardware specifications. Current model scale and performance are somewhat constrained by existing hardware conditions.

Algorithmic Aspects: MCTS search depth has room for improvement, neural network architecture may not be optimal, and randomness handling mechanisms need further refinement. These factors limit further system performance enhancement.

Application Scope: Current research primarily optimizes for Einstein Chess, requiring further validation of system generalization capabilities and adaptability in other stochastic games.

Evaluation Framework: Lacks extensive testing data from high-level human opponents, evaluation metric system requires improvement, and long-term performance stability needs additional verification.

5.3. Future Research Directions

Building upon our research achievements and limitations, future research directions can be pursued in several key areas:

In terms of algorithmic optimization, we propose exploring more efficient MCTS variants, designing optimized neural network architectures, improving randomness handling mechanisms, and investigating novel training strategies. These optimizations will contribute to enhancing the system's overall performance.

Regarding performance enhancement, efforts should focus on optimizing computational resource utilization, accelerating training processes, improving decision-making speed, and strengthening system stability. These improvements will make the system more suitable for practical applications.

For generalization capabilities, we suggest extending the system to other stochastic games, investigating knowledge transfer methodologies, exploring the possibilities of universal game AI, and enhancing system adaptability.

To improve practical applicability, we recommend developing lightweight versions, optimizing human-computer interfaces, implementing training assistance features, and incorporating analytical tools, making the system more user-friendly and deployable.

In theoretical research, we emphasize the need to deepen stochastic game theory, study algorithm convergence properties, explore more scientific evaluation methods, and perfect the theoretical framework.

For application extensions, we propose investigating educational applications, researching commercialization pathways, developing competitive platforms, and promoting practical implementations.

Future research will focus on addressing these challenges, further improving system performance, and expanding application scope, thereby contributing to artificial intelligence advancement in stochastic games. With ongoing improvements in computational capabilities and algorithmic refinements, we anticipate significant breakthroughs in this field. This research provides valuable insights for the development of stochastic game AI, and future work will continue to build upon these findings, driving technological progress and application expansion.

Acknowledgments

This work it was supported by National College Students' innovation and entrepreneurship training program project.

References

[1] CAI Biao, XU Xinyi, XIE Ting, et al. Research on improved deep neural networks in Einstein Chess[J]. Journal of Chongqing University of Technology(Natural Science), 2024, 38 (05): 108-114.

[2] XUE Yonghong, WANG Hongpeng. History and enlightenment of computer chess: From Deep Blue to AlphaZero [J]. Science & Technology Review, 2019, 37(19): 87-96.

[3] Silver D, Huang A, Maddison C, et al. Mastering the Game of Go with Deep Neural Networks and Tree Search [J]. Nature,2016, 529 (7587): 484 - 489.

[4] Silver D, Schrittwieser J, Siomonyan K,et al. Mastering the Game of Go without Human Knowledge [J]. Nature,2017,550 (7676): 354 - 359.

[5] ZHOU Zilong. Implementation and optimization of game search tree algorithms[J]. Scientific and Technological Innovation, 2021, (18): 108-110.

[6] MING Hongbing. Design and implementation of deep neural networks for Monte Carlo tree search algorithm[D]. Northeastern University, 2021. DOI:10.27007/d. cnki.gdbeu. 2021. 001565.

[7] GAO Tongtong, DING Jiahui, SHU Wenao, et al. Research on NoGo game system based on AlphaZero[J]. Intelligent Computer and Applications, 2022, 12(11): 138-141+147.

[8] ZHANG Zeyang. Research and implementation of perfect information game theory based on reinforcement learning[D]. Xidian University, 2021. DOI:10.27389/ d.cnki.gxadu. 2021. 002956.