Techno-Science Research Journal V13 (1) (2025) P 75-83

Content list available at ITC

Techno-Science Research Journal

Journal Homepage: http://techno-srj.itc.edu.kh/

Techno-Science Research Journal

CNN-based Reinforcement Learning with Policy Gradient for Khmer Chess

Both Chan [1*], Dona Valy [2], Phutphalla Kong [2]

[1] *Graduate student, Master program of Computer Science Engineering, Graduate school, Institute of Technology of Cambodia*

[2] *Research and Innovation Center, Institute of Technology of Cambodia, Russia Federation Blvd., P.O. Box 86, Phnom Penh, Cambodia*

Received: 20 August 2024; Revised: 02 September 2024; Accepted: 10 September 2024; Available online: 30 April 2025

Abstract: *Artificial intelligence, fueled by machine learning and deep learning techniques, is revolutionizing various domains. Reinforcement learning (RL) stands out as a potent method for training agents to navigate complex environments and make informed decisions. Our focus is on applying RL techniques, specifically Convolutional Neural Networks (CNNs) combined with policy gradient methods, to enhance the gameplay experience of Khmer chess. Our goal is to surpass the performance of traditional chess engines. The system employs deep neural networks to train AI agents, enabling self-play iterations for strategy refinement. Specifically, we utilize RL technology to iteratively enhance game strategies based on self-matching results, ultimately improving the system's chess proficiency. Our approach entails developing a CNN-based RL system tailored for Khmer chess, encompassing strategies, value evaluation mechanisms, and rule adaptations specific to the game. We utilize deep neural networks to facilitate agent training through self-play iterations, leveraging RL techniques for continual strategy refinement. To enhance training efficiency, we introduce a segmentation method for Khmer chess stages, optimizing the neural network's learning process by mapping game situations to optimal actions based on cumulative rewards. Furthermore, we integrate RL principles to guide action selection towards maximizing reward values, employing Deep Q-Learning with policy gradient for optimal decision-making. With the experimental validation demonstrates the efficacy of our CNN-based RL system in enhancing Khmer chess gameplay. The system exhibits self-improvement, adaptability, and human-like gameplay characteristics, enriching player experience and entertainment value. Moreover, the proposed approach showcases improved training efficiency compared to conventional RL-based chess systems, highlighting its efficacy and scalability for AI-driven game enhancements.*

Keywords: Reinforcement learning(RL), Policy gradient methods, Self-play iterations, Human-like gameplay characteristics

1. INTRODUCTION

Reinforcement learning (RL) has emerged as a pivotal area of focus in the fields of artificial intelligence (AI) and machine learning, particularly in the context of chess and other complex strategic games. This niche within AI is dedicated to creating intelligent agents capable of making sequential decisions in ever-changing environments. Unlike traditional supervised learning, which relies on labeled data to train models, RL agents learn by continuously interacting with their surroundings, receiving feedback in the form of rewards or penalties. This trial-and-error learning process enables the development of sophisticated strategies that adapt to the dynamic nature of real-world scenarios. Chess, with its intricate decision-making processes and strategic depth, serves as an excellent platform for RL, providing an ideal environment for testing and refining RL algorithms.

The conceptual origins of RL can be traced back to the field of behavioral psychology, particularly the exploration of learning through trial and error. Pioneering work in this area laid the theoretical foundations of RL, which were further developed in the mid-20th century with the formulation of dynamic programming by Richard Bellman. Bellman's work introduced key concepts such as the Bellman equation, which remains fundamental to modern RL algorithms. However, it wasn't until the advent of powerful computational resources and advanced algorithms that RL began to garner widespread attention within the domain of machine learning.

Chess, renowned for its strategic depth and historical significance, presents a formidable challenge for AI due to

* Corresponding author: Chan Both
E-mail: chanboth180720@gmail.com ; Tel: +855-10 431 168

its complexity. The game's vast branching factor, where each move leads to a multitude of possible future positions, makes it difficult for traditional chess engines to navigate the game tree efficiently. While conventional chess engines leveraging rule-based systems and minimax algorithms have achieved notable success, they encounter significant difficulties in complex endgame scenarios where precise calculation and long-term planning are essential.

The primary aim of this research is to pioneer self-play mechanisms in Khmer chess, ultimately striving towards the attainment of superhuman-level play. By enabling agents to learn and refine their strategies through iterative gameplay against themselves, the goal is to develop algorithms capable of surpassing human performance thresholds in Khmer chess.

Several pivotal works lay the foundation for this study. Notably, the AlphaZero algorithm has demonstrated the power of general reinforcement learning algorithms in mastering complex games like chess, shogi, and Go through self-play [5]. AlphaZero achieved superhuman performance without game-specific knowledge, utilizing deep neural networks and a Monte Carlo tree search (MCTS) instead of handcrafted rules. Another study applied deep reinforcement learning to finite state single-player games like Solitaire Chess [6], showcasing the efficiency and accuracy of RL models in solving puzzles by exploring fewer possible moves compared to brute force methods.

Further advancements in the field include the development of adaptable chess environments for detecting human-understandable concepts learned by RL agents [9]. This research highlights the importance of explainable AI and provides tools for research groups with limited computational resources. The use of policy gradient methods has also been explored for training neural networks [12,10], allowing for learning from a system of rewards and imposing structural constraints without complex architectures.

Recent studies have expanded the application of RL to non-traditional board games and culturally significant games, demonstrating the adaptability of RL across diverse gaming contexts. For example,[13] explored RL applications in Mancala, a traditional African game with deep cultural roots, illustrating how RL can adapt to unique strategies and rules inherent in non-Western games.[16] focused on RL in Xiangqi (Chinese Chess), a game known for its complexity and larger board size, revealing how RL can tackle culturally specific strategic challenges. Similarly,[17] investigated the use of deep RL in Hnefatafl, a historical Viking board game, showing that RL could handle asymmetric gameplay and ancient strategic elements. Moreover,[14] applied RL to Shogi (Japanese Chess), combining Policy Gradient methods with neural Monte Carlo Tree Search (MCTS) to enhance decision-making in a game with intricate piece interactions and rule sets.

These studies underscore the increasing interest in applying RL to culturally significant and non-traditional board games, highlighting the novelty and potential impact of applying similar techniques to Khmer Chess.

2. OVERVIEW OF KHMER CHESS

Khmer Chess, known locally as Ouk Chatrang [8], is a traditional Cambodian board game that shares similarities with international chess but features unique pieces and movement rules. Played on an 8x8 board, the game includes pieces such as the Neang (queen), Sdaach (king), Tuk (rook), Koul (knight), Ou (pawn), and Khon (bishop). Each piece's movement is distinct from its international counterpart, offering a unique strategic experience.

2.1. Piece Movement Rules

Each piece has unique movement capabilities that dictate how it navigates the board [18]. Understanding these movements is crucial for mastering the game. Here are the movement rules for each piece:

- Pawn (Neang): Moves one square forward to an empty square or captures one square diagonally forward. Upon reaching the eighth rank, it is promoted to a queen.
- Knight (Ses): Moves in an "L" shape (two squares in one direction and one square perpendicular) and can jump over other pieces.
- Bishop (Trun): Moves one square diagonally in any direction.
- Rook (Tuuk): Moves any number of squares horizontally or vertically, capturing by landing on an opponent's piece.
- Queen (Neang Kou): Moves one square diagonally.
- King (Sen): Moves one square in any direction, cannot move into check.

2.2. Special Opening Moves

Khmer Chess features unique opening moves that add an exciting strategic layer to the beginning of the game. These special moves are designed to enhance the dynamics of the early game. Here are the special opening moves:

- King's Jump: The king can jump like a knight to the second row on its first move, but not if in check or if blocked by an enemy rook.

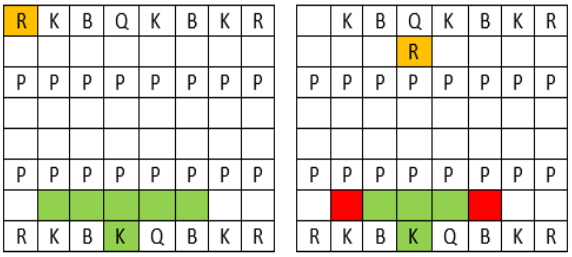

Fig. 1. King opening first move

- Queen's Jump: The queen can jump two squares forward on its first move, without capturing.

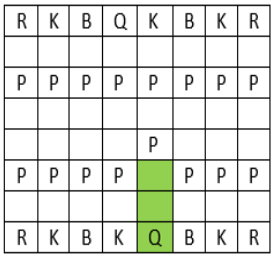

Fig. 2. Queen opening first move

2.3. Counting Rules for Draws

To prevent indefinite gameplay, Khmer Chess incorporates counting rules:

- Board's Honor Counting: Initiated when a player has three or fewer pieces. Counting starts at 1 with a limit of 64 moves. The chasing player must checkmate within this limit, or the game is drawn.
- Piece's Honor Counting: Activated when no unpromoted pawns are left, and a player has only the king. The count begins with the total number of pieces plus one, and the limit varies based on the material advantage (e.g., two rooks: 8 moves; one knight: 64 moves). Once started, the limit remains fixed regardless of subsequent captures.

These rules, reflecting the cultural depth and strategic richness of Khmer Chess, distinguish it from international chess.

3. METHODOLOGY

3.1. Status of the Khmer Chess

The status of the chess game that base of Figure 1 [3] involves a detailed assessment of the current situation and strategic developments occurring on the chessboard:

- Agent: The entity that learns to play chess through the principles of reinforcement learning. Its primary goal is to develop strategies that maximize cumulative rewards over the course of many games. The agent processes information from the game environment and makes decisions based on the current state of the game.
- Action: Involves a variety of potential moves, each of which must conform to the rules of the game. These actions can include moving a piece to a legal position, capturing an opponent's piece, promoting a pawn to a more powerful piece (such as a queen). Each action the agent takes is influenced by its evaluation of the current board state and its strategic objectives.
- State: Representation of the board at any given moment. It includes the positions and types of all pieces, the move count, and the player's turn. The state serves as the primary input to the agent's decision-making process. Accurately modeling and understanding the state is crucial, as it affects the agent's ability to predict the outcomes of its actions and plan strategically.

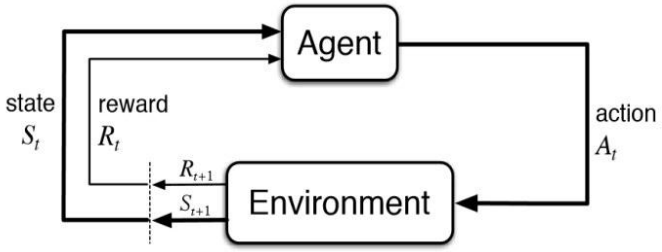

Figure 3. Reinforcement Learning Framework

- Environment: Encompasses all aspects of the chess game setup and execution. It includes the initialization of the board, enforcement of the rules to check for legal moves, and verification of game over conditions such as checkmate, stalemate, or draw. The environment is responsible for providing rewards based on the agent's actions and the game's outcome, which are essential for reinforcement learning.
- Reward: Rewards in chess are structured to guide the agent towards achieving its objectives. Positive rewards are given for favorable outcomes, such as winning the game through checkmate or capturing an opponent's piece. Conversely, negative rewards are assigned for undesirable actions, such as making illegal moves or losing pieces.

3.2. Integration of Policy Gradient Methods and Convolutional Neural Networks

By combining Policy Gradient methods with Convolutional Neural Networks (CNN), the AI agent learns to interpret chessboard states, extract relevant patterns, and probabilistically predict effective moves, thereby facilitating strategic decision-making in playing chess. This combination allows the agent to enhance its policy through experience and self-play, ultimately aiming for stronger gameplay and competitive performance.

3.2.1. Convolutional Neural Network

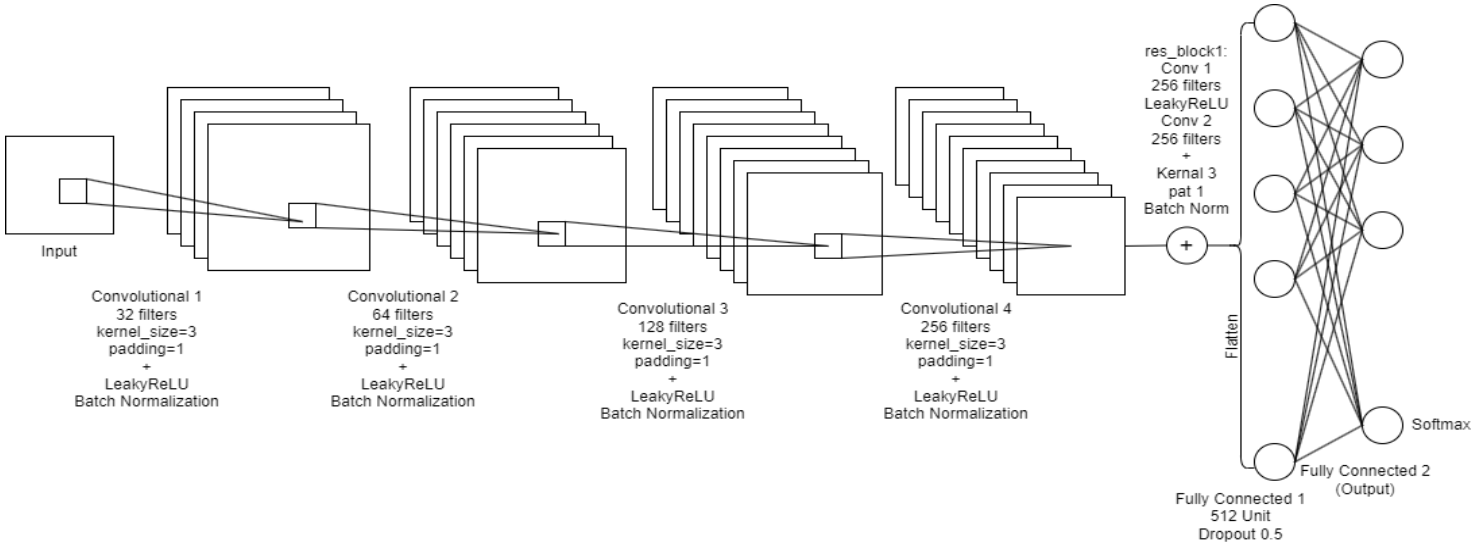

Fig. 4. The possible Convolutional Neural Network Architecture

For this research we use convolutional neural networks-based from Figure 2 for self-training policy with the Advanced Policy Network model designed for policy learning tasks, featuring a series of convolutional, batch normalization, and fully connected layers, augmented by a residual block for enhanced feature extraction. The architecture is structured as follows:

- Convolutional Layers: Four convolutional layers progressively increase the number of filters (32, 64, 128, 256) with a 3x3 kernel size and padding of 1, each followed by batch normalization and Leaky ReLU activation to stabilize training and non-linearity. The use of convolutional layers for feature extraction [1, 2].
- Residual Block: A residual block with two convolutional layers (256 filters, 3x3 kernel, padding of 1) and batch normalization layers, enhanced by Leaky ReLU activation, facilitates better gradient flow and deeper feature learning by adding the block's input to its output [7].
- Fully Connected Layers: The output from the residual block is flattened and passed through a fully connected layer with 512 neurons and Leaky ReLU activation, followed by a dropout layer (0.5 probability) to prevent overfitting [15].
- Output Layer: A final fully connected layer maps to the output size, with a softmax activation function to produce a probability distribution over the possible actions [4].

This architecture is effective for extracting intricate features from high-dimensional inputs and providing probabilistic action decisions, making it suitable for reinforcement learning .

Table 1. Hyperparameter settings

Hyperpara-meter	Value	Explanation
Discount Factor (γ)	0.9	Chosen to balance between immediate and future rewards. This value was selected based on preliminary tests showing improved performance in long-term strategic planning.
Exploration Rate (ε)	0.2	Set to encourage exploration of various strategies early in training. This rate was gradually decreased as the AI became more confident, transitioning to exploitation.
Learning Rate	0.001	Selected from a range of values tested (0.01 to 0.001) to ensure stable convergence and minimize policy loss without causing oscillations.
Batch Size	128	Chosen to balance computational efficiency and stability of gradient updates.
Replay Buffer Size	10000	Set to store a diverse set of experiences while managing memory usage effectively.

The choice of hyperparameters involved several methods:

- Empirical Testing: Initial values were chosen based on domain knowledge and preliminary testing. These values were adjusted iteratively based on observed performance in self-play and evaluation games.
- Grid Search: A grid search was employed to explore different values for key hyperparameters, such as the learning rate. This systematic approach helped identify optimal values by evaluating performance across a range of settings.
- Manual Adjustment: Parameters like the exploration rate were adjusted manually to align with observed changes in the AI's learning behavior, transitioning from exploration to exploitation as training progressed.

The final set of hyperparameters was refined through iterative testing and evaluation, focusing on performance

metrics such as win rate, draw rate, and handling of complex endgame situations.

3.2.2. Policy Gradient Method

The policy gradient method is a prominent approach in reinforcement learning (RL) that directly optimizes the policy by maximizing the expected return. Unlike value-based methods, which estimate the value function to derive the policy, policy gradient methods focus on learning a parameterized policy that dictates the agent's actions in the environment. This method is particularly useful for problems where the action space is continuous or where it is more straightforward to model the policy directly rather than the value function [10].

- **Formulation**: Let $\pi_\theta(a|s)$ denote the policy parameterized by θ, which outputs the probability of taking action a given state s. The objective is to maximize the expected return $J(\theta)$:

$$J(\theta)=E_{\pi\theta}[\sum^{T}_{t=0}\gamma^{t}R_t] \quad \text{(Eq. 1)}$$

where γ is the discount factor, and Rt is the reward at time step t.

- **Gradient Estimation**: The policy gradient theorem provides a way to compute the gradient of the expected return with respect to the policy parameters:

$$\nabla_\theta J(\theta)=E_{\pi\theta}[\nabla_\theta \log_{\pi\theta}(a|s)Q^{\pi\theta}(s,a)] \quad \text{(Eq. 2)}$$

where $Q^{\pi\theta}(s,a)$ is the action-value function under policy π_θ. In practice, the true action-value function is often approximated by the cumulative rewards observed during training [11].

- **Algorithm**: The policy gradient algorithm involves the following steps:
  - Collect Trajectories: Generate multiple episodes (trajectories) by interacting with the environment using the current policy π_θ. Each trajectory consists of a sequence of states, actions, and rewards:

$$\tau = \{(s_0, a_0, r_0), (s_1, a_1, r_1), \dots, (s_T, a_T, r_T)\} \quad \text{(Eq. 3)}$$

  - Compute Returns: For each trajectory, compute the cumulative return Rt for each time step t:

$$R_t = \sum^{T}_{k=t}\gamma^{k-t}r_k \quad \text{(Eq. 4)}$$

This cumulative return represents the total discounted reward from time step t to the end of the episode.

  - Update Policy Parameters: Use the computed returns to update the policy parameters θ via gradient ascent:

$$\theta \leftarrow \theta + \alpha\nabla_\theta J(\theta) \quad \text{(Eq. 5)}$$

where α is the learning rate, which controls the step size of the parameter update. The gradient $\nabla_\theta J(\theta)$ is estimated using the collected trajectories and the returns, as given by the policy gradient theorem.

Explanation: The policy gradient method utilizes a set of core equations to guide policy optimization effectively. Eq. 1 establishes the target function J(θ), representing the expected cumulative return by summing up discounted rewards achieved through interaction. To refine this target, Eq. 2 derives the gradient of J(θ) concerning the policy parameters θ, which directs the adjustments needed to improve the policy. The optimization process starts with gathering trajectories, detailed in Eq. 3, which capture the detailed flow of states, actions, and rewards. These trajectories are crucial for calculating the cumulative return, as shown in Eq. 4. The final step, outlined in Eq. 5, involves updating the policy parameters iteratively. This process involves adjusting the parameters in the direction of the gradient to progressively enhance the policy and maximize the expected return.

3.2.3. Training Process

The Khmer Chess board state is represented using a convolutional neural network (CNN. The CNN takes as input a tensor representation of the board, where different channels encode the presence and type of pieces for both players. The network architecture consists of several convolutional layers with batch normalization and activation functions, followed by fully connected layers that output the action probabilities combined with the policy gradient method to train the policy network, which decides the moves for Player 1.

To follow step by step on training CNN with Policy Gradient method on self-play :

- Policy Network: The policy network, a CNN, outputs a probability distribution over possible moves given the current board state. The network is trained to maximize the expected cumulative reward by selecting moves that lead to favorable outcomes.
- Self-Play for Data Generation: The training data is generated through self-play, where Player 1 uses the policy network to select actions, and Player -1 makes random moves. During self-play, the states, actions, and rewards are recorded to create a dataset for training.
- Reward Structure: Rewards are assigned based on the game outcomes and intermediate moves with the
  - Win: +10 points
  - Capture: +1 point per opponent piece captured

- Illegal Move: -1 point
- No Move: -1 point when no legal moves are available

- Policy Gradient Optimization: The policy gradient method updates the policy network by optimizing the policy loss function in Eq. 6. The loss function is defined as the negative log probability of the selected actions multiplied by the corresponding rewards:

$$\text{Policy Loss} = -\sum(\log(\pi(s, a))\cdot R) \qquad \text{(Eq. 6)}$$

The network parameters are adjusted using gradient descent to minimize the policy loss, thereby improving the policy over time.

- Replay Buffer: A replay buffer stores the state-action-reward tuples collected during self-play. The buffer allows the network to be trained on a diverse set of experiences, promoting stability and efficiency in learning with the replay buffer size 1000.
- Batch Training: The policy network is updated in batches sampled from the replay buffer. Rewards are normalized to stabilize training, and the loss is computed for the batch to update the network parameters.
- Evaluation Metrics: The performance of the policy network is evaluated based on the number of wins, losses, draws and the cumulative rewards accumulated by Player 1 and Player 2(Baseline model).

4. IMPLEMENTATION

4.1. Modeling

4.1.1. Baseline Model

The baseline model serves as a fundamental approach for simulating game moves when no advanced strategy or learning mechanism (like the policy network) is employed. It operates primarily based on random selections from the available legal moves on the chessboard.

This method is responsible for executing a move on the board based on randomness when it's the AI's turn to play with the process of:

- Retrieves all legal moves available for the current player using.
- If no legal moves are available, it penalizes the current player by decreasing their reward.
- If legal moves are available, it randomly selects one move from the list of legal moves.

Strengths of the Baseline Model:

- Simplicity: The baseline model is straightforward, requiring minimal computational resources and no complex setup, making it easy to implement and run.
- Speed: Since it relies on random selection, the baseline model executes moves quickly without the need for deep calculations or extensive training.
- Benchmarking: It provides a clear benchmark to evaluate the effectiveness of more sophisticated models. By comparing the RL model's performance against the baseline, researchers can quantify improvements and validate the benefits of advanced strategies.

The baseline model is essential for establishing a reference point in AI development. It helps identify how much the RL model improves over a basic, non-strategic approach. By starting with the baseline, researchers can clearly demonstrate the advantages of more complex methods, like policy networks, in enhancing decision-making and game performance.

4.1.2. Policy Gradient Model

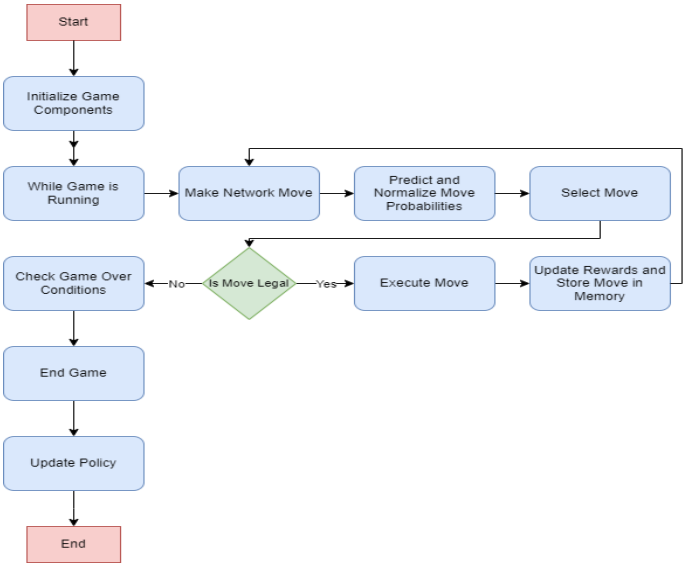

Fig. 5. Self-play Training using Policy Gradient

The policy gradient model depicted in Figure 3 involves several key stages, structured to optimize decision-making in a game-playing environment. The process can be divided into initialization, iterative gameplay, policy updates, and termination. Below is a detailed breakdown of each step:

- Initialize Game Components: Set up the game environment, policy network, and relevant parameters to prepare for training. In Khmer chess, the self-play mechanism is optimized through dynamic difficulty adjustment, where the AI progressively challenges itself

to improve learning efficiency; rule-based tweaks, incorporating specific Khmer chess rules and strategies, such as special opening moves and unique endgame conditions; and enhanced exploration, which increases the exploration rate to ensure diverse move sequences and strategic variety during training.

- While Game is Running: The model continually makes decisions based on the current game state by makes a network move and predicts the probabilities of potential moves based on the game state. These probabilities are then normalized to form a valid probability distribution. A move is selected based on the predicted probabilities.
- Check Legal Move: Ensure the selected move adheres to the game's rules. If the move is illegal, the model retries until a valid move is chosen.
- Execute Move, Update Rewards, and Store Move in Memory: After executing the move, the model updates the rewards based on the new game state. The executed move, along with the state and reward, is stored in memory for future learning.
- Check Game Over Conditions: Evaluate if the game has ended (win, loss, or draw). If so, proceed to the end game stage; otherwise, continue making moves.
- End Game and Update Policy: Upon game completion, update the policy network based on the accumulated experiences. Adjust the neural network's weights to maximize the expected reward of policy gradient methods.

4.2. Verification with Unit Test of Individual Pieces

Unit tests are crucial for verifying that each chess piece behaves according to its movement rules on an 8x8 chessboard. The Table 1 show board coordinates are defined as (x, y), where x ranges from 0 to 7 (rows from top to bottom) and y ranges from 0 to 7 (columns from left to right). Below are the unit tests for each piece, specifying expected outcomes and conditions:

Table 2. Unit Test of chess pieces

Chess Piece	Testing Moves	Expected Result	Condition
Bishop	(7,2) to (6,3)	True	Diagonal move
	(7,2) to (6,2)	True	Forward move
	(7,2) to (5,4)	False	Invalid move
King	(7,3) to (6,3)	True	Single-step vertical move
	(7,3) to (6,2)	True	Single-step diagonal move
	(7,3) to (5,4)	False	Invalid move
Knight	(7,1) to (5,0)	True	L-shape move
	(7,1) to (6,1)	False	Invalid move
Pawn	(5,0) to (4,0)	True	Forward move for player 1
	(5,0) to (3,0)	False	Invalid double move
	(2,0) to (3,0)	True	Forward move for Player 2
	(2,0) to (4,0)	False	Invalid double move
Queen	(7,4) to (6,5)	True	Diagonal move
	(7,4) to (5,6)	False	Invalid move
	(7,4) to (4,4)	False	Invalid move
Rook	(7,0) to (6,0)	True	Vertical move
	(7,0) to (7,1)	True	Horizontal move
	(7,0) to (6,1)	False	Invalid move

These unit tests ensure that each piece's movement adheres to the established rules of KhmerChess. Each test case provides a clear validation of the piece's move is legal or not, contributing to the overall reliability and accuracy of the chess engine's implementation.

5. RESULTS AND DISCUSSION

Applying Reinforcement Learning (RL) to Khmer chess involves closely examining how AI agents perform and behave as they're trained using RL algorithms. The results of this process depend on various factors, such as how the training is structured, which algorithm is used, and how complex the Khmer chess problem is. The main goal is to teach the AI agent all the rules of the game and help it become skilled at making the best moves.
To effectively evaluate the performance of these models, specific metrics are employed, as shown in Table 2:

- Win Rate: This shows how often the AI agent wins games against other baseline AI opponents.
- Average Game Length: This tells us the average number of moves it takes for the AI agent to win or lose a game within 200 moves.
- Time Efficiency: This measures how quickly the AI agent makes moves during a game, usually in seconds per move.

These evaluation metrics give us numbers that help us understand how well the deep learning model trained with policy gradient methods is performing.

Table 3. Illustrates the results of the gameplay metrics

Games	1-2000	2001-4000
Policy Gradient Win Rate	18%	20%
Baseline Win Rate	17%	18%
Draw Rate	65%	62%
Average Game length	160.56 moves	160.30 moves

The results indicate a gradual improvement in the win rate as the number of training games increases. The win rate for the policy gradient method starts at 18% after 1 to 2000 games and rises to 20% by the 4000th game, demonstrating the positive impact of accumulated experience on the model's performance. The draw rate shows a slight decrease from 65% to 62% because the model is progressively adopting more effective strategies and making more decisive moves, reducing the number of games that end in a draw as its gameplay improves with accumulated experience, indicating a consistent performance in achieving game outcomes that are neither wins nor losses.

Additionally, the average game length decreases marginally from 160.56 moves to 160.30 moves, suggesting a trend towards more intricate gameplay as training progresses. The move time remains stable at 0.01 seconds, reflecting the model's consistent computational efficiency throughout the training process.

6. CONCLUSIONS

In conclusion, this study began by recognizing the inherent complexities of Khmer chess, characterized by intricate move dynamics and a vast game tree. Traditional rule-based systems have significant limitations in addressing the dynamic and uncertain nature of Khmer chess gameplay, particularly as positions evolve. The challenges of strategic long-term planning, adapting to varied playing styles, and efficiently managing the game tree complexity were pivotal considerations that spurred the investigation of reinforcement learning (RL) as a potential solution. The successful integration of RL in Khmer chess situates this research at the forefront of innovative AI applications in strategic board games. The insights gained provide a foundation for further exploration and refinement of RL techniques, opening new horizons for the development of intelligent agents capable of mastering the intricacies of culturally significant games like Khmer chess. The primary findings of this research include the identification of the game's complexity and the limitations of traditional systems, the successful application of RL, and the establishment of a foundation for future research. To enhance the AI's performance, future work will focus on collecting data from human play and integrating it with reinforcement learning. This approach aims to help the agent learn all the game rules and become skilled at making optimal moves. Initially, the AI will explore the game space randomly, akin to human players seeking strategic insights. As training progresses, the AI will become more proficient, relying less on random moves and more on learned patterns and strategies derived from human playstyles stored in its neural networks. Combining RL with insights from human gameplay is expected to accelerate the AI's learning curve and enable it to adapt more effectively to varied playing styles. This hybrid approach is anticipated to result in a more robust and sophisticated AI, capable of navigating the complexities of Khmer chess with greater finesse and strategic depth.

REFERENCES

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet classification with deep convolutional neural networks," Adv. Neural Inf. Process. Syst., 2012, pp. 1097-1105.

[2] A. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv, 2014. [Online]. Available: https://arxiv.org/abs/1409.1556. [Accessed: Sept. 10, 2024].

[3] A. Anwar, "Basic terminologies of reinforcement learning," Analytics Vidhya. [Online]. Available: https://medium.com/analytics-vidhya/basic-terminology-reinforcement-learning-2357fd5f0e51. [Accessed: Sept. 10, 2024].

[4] C. M. Bishop, Pattern Recognition and Machine Learning, 1st ed. New York, NY, USA: Springer, 2006. https://arxiv.org/pdf/2211.05500

[5] D. Silver, T. Hubert, and J. Schrittwieser, "A general reinforcement learning algorithm that masters chess, shogi and Go through self-play," arXiv preprint arXiv, 2017. [Online]. Available: https://arxiv.org/pdf/1712.01815v1. [Accessed: Sept. 10, 2024].

[6] D. Stephens, "Applying Deep Reinforcement Learning to Finite State Single Player Games," Stanford University. [Online]. Available: https://cs229.stanford.edu/proj2019aut/data/assignment_308832_raw/26641389.pdf. [Accessed: Sept. 10, 2024].

[7] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., Las Vegas, NV, USA, 2016, pp. 770-778.

[8] Ouk Chaktrang Championship, "Counting rule, strategies, tactics," Pre-SouthEast Asia Game 32nd 2023 in Cambodia. [Online]. Available: https://docs.google.com/document/d/1adppJ66vonM27UYwC-KyldXl7oZ_5Pb0/edit. [Accessed: Sept. 10, 2024].

[9] P. Hammersborg and I. Strumke, "Reinforcement learning in an adaptable chess environment for detecting human-understandable concepts," arXiv preprint arXiv, 2022. [Online]. Available: https://arxiv.org/pdf/2211.05500. [Accessed: Sept. 10, 2024].

[10] R. S. Sutton, D. A. McAllester, S. P. Singh, and Y. Mansour, "Policy gradient methods for reinforcement learning with function approximation," Adv. Neural Inf. Process. Syst., 2000, pp. 1057-1063.

[11] R. J. Williams, "Simple statistical gradient-following algorithms for connectionist reinforcement learning," Mach. Learn., vol. 8, no. 3, pp. 229-256, May 1992.

[12] S. Bose, "Training neural networks with policy gradient," ResearchGate. [Online]. Available: https://www.researchgate.net/publication/316171194_Training_Neural_Networks_with_Policy_Gradient. [Accessed: Sept. 10, 2024].

[13] S. Firoozshahian and E. Mazlumian, "Reinforcement learning in Mancala: Adapting machine learning techniques to traditional African games," Int. J. Game Theory, 2020.

[14] S. Lee, H. Kim, and Y. Takahashi, "Enhancing Shogi AI with policy gradient and neural MCTS: A reinforcement learning approach," IEEE Trans. Games, vol. 15, no. 2, pp. 199-210, Apr. 2023.

[15] S. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, no. 1, pp. 1929-1958, Jun. 2014.

[16] X. L. Xu and X. Yao, "Reinforcement learning in Chinese Chess: A study of AI development in culturally significant games," J. Artif. Intell. Res., 2021.

[17] Y. Zhang, M. Wang, and P. Li, "Deep reinforcement learning for Hnefatafl: Addressing asymmetric strategy in historical Viking board games," J. Mach. Learn. Res., vol. 23, pp. 1-30, Apr. 2022.

[18] "Ouk Chaktrang" PyChess, 2022. [Online]. Available: https://www.pychess.org/variants/cambodian. [Accessed: Sept. 10, 2024].