MendikotZero: An AlphaZero-Inspired Reinforcement Learning Agent for Imperfect-Information Card Games

Bhushan Ladgaonkar
Independent Researcher
Mumbai, India
bhushanladgaonkar@gmail.com

***Abstract*—This paper details the design, implementation, and training of MendikotZero, an autonomous Artificial Intelligence agent capable of playing the Indian trick-taking card game *Mendikot* at a high strategic level. Unlike traditional board games with perfect information, Mendikot presents unique challenges involving hidden information, stochasticity, and partnership-based cooperation. The proposed agent employs a Deep Reinforcement Learning (DRL) methodology inspired by the AlphaZero framework, utilizing a Dual-Head Neural Network to guide a Monte Carlo Tree Search (MCTS) algorithm.**

The AI was trained entirely from scratch via self-play without prior human knowledge or heuristics. Performance analysis of saved checkpoints demonstrates a distinct evolution in gameplay strategy: transitioning from stochastic, reactive play to sophisticated, long-horizon planning. The final model exhibits advanced behaviors including resource management (preserving high-value cards), dynamic trump control, and risk mitigation, validating the efficacy of MCTS approaches in complex combinatorial games with imperfect information.

***Index Terms*—Deep Reinforcement Learning, Monte Carlo Tree Search (MCTS), AlphaZero, Game Theory, Imperfect Information, Neural Networks.**

I. Introduction

Trick-taking card games represent a fascinating frontier for artificial intelligence research. Unlike Chess or Go, where the game state is fully visible to all players (Perfect Information), card games like Bridge and Mendikot involve hidden hands (Imperfect Information), requiring agents to reason under uncertainty and infer probabilities based on opponent actions.

This project focuses on **Mendikot** (also known as *Mindi*), a popular partnership-based game from the Indian subcontinent. The primary objective was to develop an agent, *MendikotZero*, capable of learning winning strategies purely through self-play mechanisms.

A. The Game of Mendikot

Mendikot is played by four players in two fixed partnerships using a standard deck (often with 2s removed, resulting in 48 cards). The objective is to capture tricks containing specific high-value cards—the Tens (10s), referred to as "Mendis." Key challenges for an AI include:

- **Imperfect Information:** Players observe only their hand and played cards. The distribution of the remaining cards must be inferred.
- **Cooperative Play:** The reward signal is shared between partners. An optimal move for an individual might be suboptimal for the team.
- **Dynamic Strategy:** In the variation implemented, the "Trump Suit" is not fixed initially but is declared dynamically by the first player unable to follow the lead suit. This adds a layer of pivotal, game-altering decision-making.

B. Project Evolution

The development of MendikotZero was iterative. Initial experiments utilized a reactive Actor-Critic (A2C) architecture. However, analysis revealed significant limitations in foresight and tactical planning. To address this, the architecture was overhauled to implement a search-based approach using **Monte Carlo Tree Search (MCTS)**, guided by a deep neural network, closely mirroring the principles of the AlphaZero paper [1].

II. Methodology: Building MendikotZero

The system comprises three core pillars: a rule-rigorous game simulation environment, a sophisticated learning architecture, and a continuous self-play training loop.

A. The Game Environment

The foundation of the project is a custom Python-based simulation encapsulated in a `GameState` class. This environment ensures strict adherence to game rules and provides the interface for the agent to interact with the game world. Key responsibilities include:

- **State Management:** Tracking the deck, player hands, trick history, and suit voids (knowing which player is out of a specific suit).
- **Rule Enforcement:** validating moves, specifically the constraint that the first four tricks must ideally be of different suits if possible (in specific variations), and standard follow-suit rules.
- **Trump Logic:** Automatically detecting dynamic trump declaration and calculating trick winners.
- **Cloning:** A critical `clone()` method allows the MCTS to create independent copies of the game state to simulate future trajectories without affecting the live game.

B. The Agent's Architecture

The agent combines the intuition of a Neural Network with the look-ahead capability of MCTS.

1) The Neural Network (The "Intuition"): The "brain" of the agent is a Deep Neural Network (DNN) with a shared body and two specialized output heads:

- **Policy Head:** Outputs a probability distribution over all legal moves. It represents the agent's instinct—answering, *"What are the most promising moves in this position?"*
- **Value Head:** Outputs a scalar value between -1 (Loss) and +1 (Win). It represents positional awareness—answering, *"Who is currently winning based on the cards played?"*

2) Monte Carlo Tree Search (The "Planning"): While the network provides intuition, MCTS provides conscious reasoning. For every decision, the agent performs hundreds of simulations to build a search tree. The process follows four phases:

1) **Selection:** The algorithm traverses the current tree using a variant of the Upper Confidence Bound for Trees (UCT) formula, balancing exploration (trying new moves) and exploitation (using known good moves).
2) **Expansion:** Upon reaching a leaf node, the Neural Network is queried to evaluate the position and generate probabilities for child nodes.
3) **Simulation/Evaluation:** Instead of playing a random game to the end (classic MCTS), the Value Head provides an immediate evaluation of the state, significantly speeding up the search.
4) **Backpropagation:** The value is propagated up the tree, updating the visit counts and win-rates of all nodes along the path.

To ensure robust exploration during training, Dirichlet Noise was added to the root node's priors, preventing the agent from becoming deterministic too early.

C. The Training Process (Self-Play)

The agent learns entirely tabula rasa (from a blank slate). Four instances of the current best model play against each other.

- **Data Generation:** Games are played using MCTS-based moves. The state, the search probabilities (policy), and the final game outcome (reward) are stored in a Replay Buffer.
- **Optimization:** The Neural Network is trained on batches from this buffer to minimize the error between its predicted policy/value and the actual MCTS results/game outcomes.

III. Results and Key Learnings

The agent was trained on local hardware (Intel i3-9100F CPU) for over 13,000 episodes. Despite limited compute resources compared to industrial labs, the agent demonstrated significant strategic convergence.

A. Performance Analysis

The training progression revealed distinct evolutionary phases:

1) **The Reckless Phase (Ep 0-4000):** The agent played randomly, often wasting high-value cards (Aces/Kings) on low-value tricks.
2) **The Grinder Phase (Ep 4000-9000):** The agent learned defensive play and risk mitigation, achieving a break-even win rate against random baselines.
3) **The Veteran Phase (Ep 9000+):** The agent began demonstrating complex behaviors, such as hoarding high cards to capture "Mendis" late-game and effectively managing the trump suit.

Fig. 1 illustrates the reward distribution of the final model (`ep_13000`) over 500 evaluation games. The distinct peaks at high positive rewards indicate that the agent is not just winning, but winning decisively by capturing the majority of point-scoring cards.

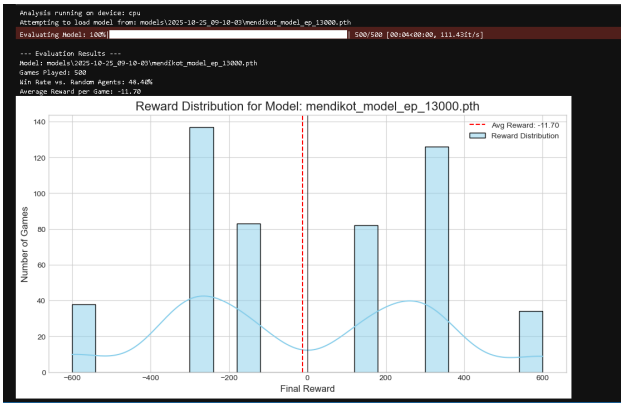

Fig. 1. Reward distribution of the ep_13000 model over 500 evaluation games. The skew towards positive rewards demonstrates the agent's ability to consistently execute high-value winning strategies.

B. Key Learnings

- **Algorithm Suitability:** The transition from Actor-Critic to AlphaZero/MCTS was the turning point. In trick-taking games, the sequence of moves is critical; MCTS allows the agent to visualize these sequences, whereas reactive models failed to look ahead.
- **State Representation:** A rich state representation—specifically one that includes a perfect history of played cards (card counting)—was essential. The agent effectively learned to "count cards" via its input vector.

IV. Discussion and Future Work

While the current system demonstrates strong play, two key areas are identified for future research:

- **Coordinated Evaluation:** Mendikot is a partnership game. Future work should focus on implementing an evaluation metric where the AI team shares a single MCTS search tree during inference to simulate explicit signal-passing and cooperative planning.

- **Competitive League Training:** Establishing a dynamic league where new versions of the agent must defeat previous checkpoints to be promoted, ensuring continuous improvement and preventing cyclic strategies.

V. Conclusion

This project successfully demonstrates the application of the AlphaZero paradigm to *Mendikot*. By leveraging self-play and MCTS, **MendikotZero** evolved from random noise to a sophisticated strategic agent capable of resource management and tactical planning. This work serves as a practical case study in applying modern Reinforcement Learning techniques to solve complex combinatorial games with imperfect information and limited compute resources.

Acknowledgment

The author would like to acknowledge **Google Gemini 2.5 Pro**, which was utilized as a technical accelerator during the development of this project. The LLM provided assistance in debugging complex MCTS logic and optimizing PyTorch implementation details. All architectural decisions, core logic, and final codebase verification were performed manually by the author to ensure reproducibility and integrity.

References

[1] D. Silver, T. Hubert, J. Schrittwieser, et al., "A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play," *Science*, vol. 362, no. 6419, pp. 1140-1144, 2018.