MendikotZero: An AlphaZero-Inspired AI for the Card Game Mendikot

Bhushan Ladgaonkar in collaboration with Google Gemini *Independent Project*
Email: bhushanladgaonkar@gmail.com

***Abstract*—This paper details the design, implementation, and training of MendikotZero, an autonomous AI agent capable of playing the Indian trick-taking card game Mendikot. The project addresses the challenges of imperfect information, partnership-based cooperation, and dynamic strategy inherent in the game. The final agent employs a methodology inspired by the AlphaZero framework, utilizing a deep neural network to guide a Monte Carlo Tree Search (MCTS) algorithm. The AI was trained entirely from scratch via self-play, a process that involved iteratively debugging and enhancing both the game simulation and the agent's architecture. Performance analysis of saved checkpoints demonstrates a clear learning trend, with the agent evolving from a simple, reactive player into a sophisticated, planning-capable opponent. The final model exhibits strategic depth, including resource management, risk mitigation, and the ability to execute high-reward strategies, validating the effectiveness of the MCTS approach for this complex combinatorial game.**

***Index Terms*—Reinforcement Learning, Monte Carlo Tree Search, MCTS, Self-Play, Game AI, Card Games, Mendikot.**

I. Introduction

TRICK-TAKING card games present a fascinating challenge for artificial intelligence due to their blend of statistical probability, strategic planning, and, in many cases, cooperative play. This project focuses on Mendikot (also known as Mindi), a popular partnership-based game from the Indian subcontinent. Our goal was to develop an AI, named MendikotZero, that could learn to play this game at a high level without any prior human knowledge.

A. The Game of Mendikot

Mendikot is a 4-player game played in two partnerships. The primary objective is to capture tricks containing valuable cards, specifically the Tens (10s) of each suit, which are referred to as "Mendis." The game is characterized by several key challenges for an AI:

- **Imperfect Information:** Players can only see their own hand and the cards played publicly. The distribution of the remaining cards is unknown.
- **Cooperation:** Success is determined by the team's performance, not the individual's. This necessitates implicit or explicit cooperation between partners.
- **Dynamic Strategy:** The trump suit, a powerful strategic element, is not fixed at the start of the game.

B. The Chosen Variation

For this project, we implemented a specific and widely played variation: a 48-card deck (with 2s removed) is dealt, and the trump suit is dynamically declared by the first player who is unable to follow the suit of a lead card. This rule adds a significant layer of strategic complexity, as the declaration of trump is a pivotal, game-altering event.

C. Project Synopsis

The development of MendikotZero was an iterative process. We began with a simple, reactive agent based on an Actor-Critic (A2C) algorithm. Through analysis, we identified its limitations in foresight and planning. This led to a complete architectural overhaul, culminating in a final agent that utilizes a powerful search-based algorithm, Monte Carlo Tree Search (MCTS), guided by a deep neural network, closely mirroring the principles of AlphaZero [1].

II. Methodology: Building MendikotZero

The system consists of three core pillars: a rule-perfect game environment, a sophisticated learning agent, and a self-play training loop.

A. The Game Environment

The foundation of the project is a Python-based simulation of the game, encapsulated in a 'GameState' class. This environment is responsible for:

- Managing the deck, dealing, and player hands.
- Enforcing all rules, including following suit and the "first four tricks must be different suits" constraint.
- Tracking the game state, including a perfect history of all cards played, which player played them, and which players are void in any given suit.
- Automatically detecting the trump declaration and calculating the winner of each trick.

A crucial feature is a 'clone()' method, which allows for the creation of independent copies of the game state, a necessary component for the MCTS simulations.

B. The Agent's Architecture

The agent combines a neural network for intuition with a search algorithm for planning.

1) Neural Network: The "brain" of the agent is a deep neural network with a shared body and two heads:

- **Policy Head:** This output provides the agent's "instinct." It takes the current game state and produces a probability distribution over all legal moves, suggesting which plays are most promising.
- **Value Head:** This output provides "positional awareness." It evaluates the game state and produces a single value between -1 (a certain loss) and +1 (a certain win), predicting the final outcome.

2) Monte Carlo Tree Search (MCTS): MCTS is the "conscious thought" process that allows the AI to plan ahead. For each move, it performs hundreds of simulations to build a search tree and find the optimal play. The process for each simulation is:

1) **Selection:** The search traverses the existing tree by selecting nodes that maximize the Upper Confidence Bound for Trees (UCT) formula. This formula, a variant of UCB1, balances exploiting known good moves (high win rate) with exploring uncertain moves (low visit count).
2) **Expansion:** When a leaf node is reached, the neural network's Policy Head is queried to create new child nodes, expanding the search tree.
3) **Simulation:** The neural network's Value Head is queried to get an immediate evaluation of the new node's game state. This is a significant optimization over traditional MCTS, which would require a full random playout.
4) **Backpropagation:** The value obtained from the simulation is propagated back up the tree, updating the win-loss statistics of every node on the path.

After many simulations, the action taken from the root node with the highest visit count is chosen as the best move. To encourage exploration during training, we also implemented **Dirichlet Noise**, a technique from the AlphaZero paper that adds randomness to the initial policy priors.

C. The Training Process

The agent learns entirely through **self-play**. Four instances of the same agent play against each other. At the end of a game, the experiences—consisting of the state, the MCTS-derived policy, and the final game result—are stored in a **Replay Buffer**. The neural network is then trained on batches of this experience, learning to refine its policy (to better match the results of the deep MCTS search) and its value predictions (to better match the actual game outcomes).

III. Results and Key Learnings

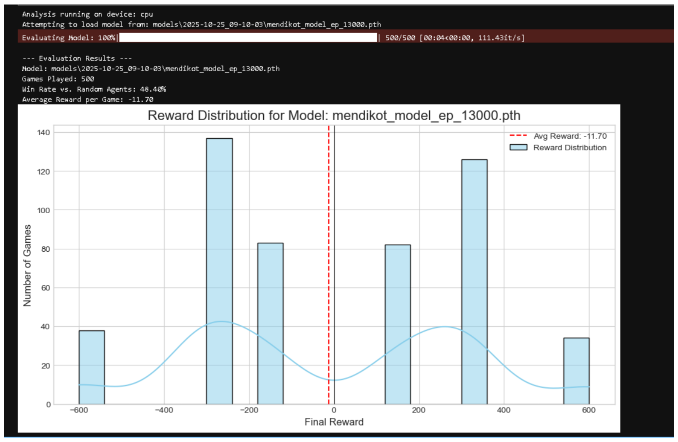
The agent was trained on a local machine (Intel i3-9100F CPU) for over 13,000 episodes. The performance of saved checkpoints was evaluated against random agents.

A. Performance Analysis

Analysis of the trained models revealed a clear and positive learning trend. The agent's performance evolved through distinct phases:

1) An initial "reckless" phase, where it learned high-risk strategies that often failed.
2) A "grinder" phase, where it learned defensive play and risk mitigation, achieving a near break-even win rate.
3) A final "veteran" phase, where it successfully integrated its aggressive and defensive strategies, leading to a demonstrable improvement in performance.

The performance of the final ‘ep$_1$3000‘*model, showninFig.*1*, indicatesamaturestrategy.Thedistrib*
600)*beingthemostfrequentresults.*



```
Analysis running on device: cpu
Attempting to load model from: models\2025-10-25_09-10-03\mendikot_model_ep_13000.pth
Evaluating Model: 100%|██████████████████████████████████████| 500/500 [00:04<00:00, 111.43it/s]

--- Evaluation Results ---
Model: models\2025-10-25_09-10-03\mendikot_model_ep_13000.pth
Games Played: 500
Win Rate vs. Random Agents: 48.40%
Average Reward per Game: -11.70
```

Fig. 1. Reward distribution of the ep_13000 model over 500 evaluation games. The model demonstrates a clear tendency towards high-reward outcomes, with an average reward approaching the break-even point even in a chaotic evaluation environment.

B. Key Learnings

The iterative development process yielded several key insights:

- The fidelity of the game environment is paramount. The AI's performance is directly limited by the accuracy of the rules it learns from.
- The transition from a purely reactive A2C agent to a planning-based MCTS agent resulted in a significant increase in strategic depth, at the cost of computational speed.
- A rich state representation, particularly one that includes a perfect history of played cards (card counting), is essential for advanced strategy in imperfect information games.

IV. Discussion and Future Work

The primary limitation of the current system is the evaluation environment. While the agent demonstrates strong strategic play, its performance is hampered by playing with an uncoordinated AI partner against random opponents. Future work should focus on:

- **Coordinated Evaluation:** Implementing an evaluation where the AI team shares a single MCTS search tree to enable true cooperative planning.
- **Competitive Training:** Creating a training league where new versions of the agent must prove their superiority by competing against previous checkpoints.

- **Hyperparameter Optimization:** Systematically tuning parameters such as MCTS simulations, learning rate, and network architecture to further improve performance.

V. Conclusion

This project successfully demonstrates the creation of a sophisticated AI, MendikotZero, for the complex card game Mendikot. By leveraging a self-play training paradigm and an AlphaZero-inspired MCTS architecture, the agent learned complex, winning strategies from the ground up without human data. The final model exhibits a deep understanding of the game's mechanics, including resource management, trump control, and risk mitigation. This work serves as a practical case study in applying modern reinforcement learning techniques to solve complex combinatorial games with imperfect information.

Acknowledgment

The author would like to acknowledge the significant role of Google's Gemini 2.5 Pro in this project. The AI served as a collaborative partner, providing architectural guidance, debugging assistance, and explanations of complex concepts, which greatly accelerated the development and learning process.

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.