Monte Carlo Tree Search Implementation for 3D Tic-tac-toe (4x4x4)

Introduction

This report details the implementation of Monte Carlo Tree Search (MCNT) algorithm for a 3D version of Tic-tac-toe played on a 4x4x4 grid. The implementation aims to create an AI agent capable of learning optimal playing strategies through repeated simulations and value function approximation.

Game Environment

Board Structure

- 4x4x4 three-dimensional grid
- Total of 64 possible positions (4 × 4 × 4)
- Two players: P1 (value: 1) and P2 (value: -1)
- Empty cells represented by 0

Winning Conditions

The game considers the following winning patterns:

1. Four consecutive marks in any horizontal row (16 possible)
2. Four consecutive marks in any vertical column (16 possible)
3. Four consecutive marks in any layer (16 possible)
4. Four consecutive marks in any diagonal within a plane
5. Four consecutive marks in any 3D diagonal across layers

MCNT Implementation Methodology

State Representation

1. Board State
    - Represented as a 3D numpy array (4×4×4)
    - Each cell contains 1 (P1), -1 (P2), or 0 (empty)
    - State hash created by flattening the 3D array to a string

2. Game State Management

   - Tracks current player's turn
   - Maintains game ending conditions
   - Records available positions for moves

Learning Process

Initialization

- Two agents (P1 and P2) created with:
  - Exploration rate (epsilon = 0.3)
  - Learning rate (alpha = 0.2)
  - Discount factor (gamma = 0.3)
  - Empty state-value dictionary

Training Iteration

Each training episode follows these steps:

1. State Selection

   - Current player evaluates board state
   - Available positions determined
   - Action selected based on epsilon-greedy strategy

2. Action Selection

   - Exploration (Random Action):
     - Probability: epsilon
     - Randomly select from available positions
   - Exploitation (Best Action):
     - Probability: 1 - epsilon
     - Choose position with highest stored state value
     - If no stored value, initialized to 0

3. State Update Process

   - Selected action applied to board
   - New state hash generated
   - State added to agent's history

- Player symbol switched

4. Reward Distribution
   When game ends:
   - Win: Reward = 1
   - Loss: Reward = 0
   - Draw: Reward = 0.5

5. Value Function Update
   For each state in reversed order:
   - Apply MCNT update formula:
     $V(S_t) \leftarrow V(S_t) + 1/N(S_t) * (G_t - V(S_t))$
   - Gt represents future rewards
   - Discount factor applied to future rewards
   - N(St) represents state visit count

Policy Storage

- State-value pairs saved to file
- Allows for policy reuse in future games
- Enables human vs AI gameplay

Performance Analysis

The implementation was tested through multiple training rounds:

1. Training Metrics
   - Win rate: 51.0%
   - Draw rate: 0.0%
   - Loss rate: 49.0%

2. Strategy Development
   - Agent learns to block opponent's winning moves
   - Develops understanding of 3D diagonal threats
   - Shows preference for center and corner positions

Conclusions

The MCNT implementation demonstrates effective learning of 3D Tic-tac-toe strategies. The balanced win rate suggests the algorithm has developed a competitive playing style, though there remains room for improvement through:

1. Parameter tuning
2. Extended training periods
3. Enhanced exploration strategies
4. Improved value function approximation

The implementation successfully handles the complexity of 3D space while maintaining reasonable computational requirements, making it suitable for both training and real-time play against human opponents.